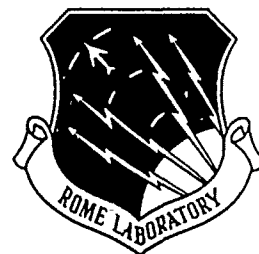


AD-A240 668



RL-TR-91-163
Final Technical Report
August 1991

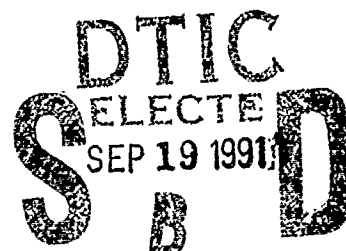


2

TESTER INDEPENDENT SUPPORT SOFTWARE SYSTEM (TISSS)

Harris Corporation

Dr. Robert Rolfe, Amy Pierce, David Lehtonen



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

91-10805



Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

9 1 9 17 015

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

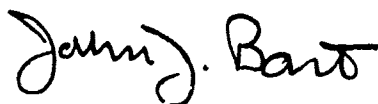
RL-TR-91-163 has been reviewed and is approved for publication.

APPROVED:



WILLIAM E. RUSSELL, JR.
Project Engineer

FOR THE COMMANDER:



JOHN J. BART
Technical Director
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(ERD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE August 1991		3. REPORT TYPE AND DATES COVERED Final Sep 85 - Jun 89	
4. TITLE AND SUBTITLE TESTER INDEPENDENT SUPPORT SOFTWARE SYSTEM (TISSS)				5. FUNDING NUMBERS C - F30602-84-C-0168 PE - 62702F PR - 3003 TA - AT WU - F1	
6. AUTHOR(S) Dr. Robert Rolfe, Amy Pierce, David Lehtonen					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Harris Corporation Attn: GASD Melbourne FL 32902				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) OUSD R&AT (CET) Rome Laboratory (ERD) Wash DC 20301 Griffiss AFB NY 13441-5700				10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-91-163	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: William E. Russell, Jr./ERD/(315) 330-3974					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report defines the objectives and technical accomplishments of the Tester Independent Support Software System (TISSS) software development effort. The objectives of this software development effort were to define, develop, and implement a software system for the automated generation and maintenance of electrical test specifications and test programs for complex Very High Speed Integrated Circuits (VHSIC) and other Very Large Scale Integrated Circuits (VLSI) devices. This was accomplished through the development of the TISSS Software system and necessary information representation languages. The TISSS provides a capability for the Government to capture design data in a standardized electronic form and use this information in a standard, transportable, and computer-accessible format and to automatically generate Automatic Microcircuit Test Equipment (AMTE) test programs for lifecycle support of VHSIC and VLSI devices.					
14. SUBJECT TERMS ANTE, ATE, Test Programs, Test Specifications, Test Program Generation, CAD to Test Link, Digital Simulation, Automation Technology				15. NUMBER OF PAGES 140	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR		

CONTENTS

1	INTRODUCTION	1
1.1	TISSS OBJECTIVE	1
1.2	BACKGROUND	2
2	TECHNICAL ACCOMPLISHMENTS	3
2.1	TISSS SOW Task (4.1.1) - Review "Pertinent Research Topics"	3
2.2	TISSS SOW Task (4.1.2) - Investigate Existing Software Packages	3
2.3	TISSS SOW Task (4.1.3) - Design TISSS Architecture	7
2.4	TISSS SOW Task (4.1.4) - Evaluate Hardware And Software	11
2.5	TISSS SOW Task (4.1.5) - Evaluate Acceptability	11
2.6	TISSS SOW Task (4.1.6) - Perform A Phase I Preliminary Design	12
2.7	TISSS SOW Task (4.1.7) - Conduct A Technical Presentation	13
2.8	TISSS SOW Task (4.1.8) - Prepare A Technical And Cost Proposal	14
2.9	TISSS SOW Task (4.2.1) - Perform A Phase II Preliminary Design	14
2.10	TISSS SOW Task (4.2.2) - Conduct A Detailed Design	17
2.11	TISSS SOW Task (4.2.3) - Code And Test Each Software Unit	20
2.12	TISSS SOW Task (4.2.4) - Integrate And Test Software	20
2.13	TISSS SOW Task (4.2.5) - Verify TISSS Operation	21
2.14	TISSS SOW Task (4.2.6) - Configuration Management Program	21
2.15	TISSS SOW Task (4.2.7) - Software Quality Assurance Program	23
2.16	TISSS SOW Task (4.2.8) - Implement Planning And Control Procedures	24
2.17	TISSS SOW Task (4.2.9) - Coordinate TISSS With IDAS/VHDL Efforts	30
2.18	TISSS SOW Task (4.2.10) - Incorporate Fault Simulators Into TISSS	33
2.19	TISSS SOW Task (4.2.11) - TISSS Interface Manual For Other CAD And Other AMTE	34
2.20	TISSS SOW Task (4.2.12) - Validate And Verify All Software	35
2.21	TISSS SOW Task (4.2.13) - Validate Adequacy And Accuracy Of Generated TIDB's And Test Programs For New Devices	36
2.22	TISSS SOW Task (4.2.14) - Validation Test Suites	37
2.23	TISSS SOW Task (4.2.15) - Generate Test Specifications And Programs	38
2.24	TISSS SOW Task (4.2.16) - Training Course	38
2.25	TISSS SOW Task (4.2.17) - Install And Demonstrate TISSS At RADC	40
2.26	TISSS SOW Task (4.2.18) - Beta Site Program (Option)	41

2.27	TISSS SOW Task (4.2.19) - Preliminary And Final Technical Brochure	41
2.28	TISSS SOW Task (4.2.20) - Coordinate With ERADCOM Contractor	41
2.29	TISSS SOW Task (4.2.21) - Provide IGES Capability	42
2.30	TISSS SOW Task (4.2.22) - Provide SQL Graph Software	42
2.31	TISSS SOW Task (4.2.23) - Define ECLIPSE OCD And S/SS	42
2.32	TISSS SOW Task (4.2.24) - Define ECLIPSE Exchange Standards	43
2.33	TISSS SOW Task (4.2.25) - Define ECLIPSE Functional Requirements	43
2.34	TISSS SOW Task (4.2.26) - VHSIC TISSS Evaluation	44
2.35	TISSS SOW Task (4.2.27) - Evaluate Tools For ECLIPSE	44
2.36	TISSS SOW Task (4.2.28) - Evaluate Host Environments	44
2.37	TISSS SOW Task (4.2.29) - Specify ECLIPSE System Requirements	44
2.38	TISSS SOW Task (4.2.30) - ECLIPSE Concept Exploration Technical Report	45
2.39	TISSS SOW Task (4.2.31) - Coordinate With Other ECLIPSE Contractor	45
2.40	TISSS SOW Task (4.2.32) - ECLIPSE Insertion Plan	45
2.41	TISSS SOW Task (4.3) Reviews	45
2.42	TISSS SOW Task (4.3.1) - Phase I Kick-off Meeting	45
2.43	TISSS SOW Task (4.3.2) - Phase I Preliminary Specification Review (PSR)	46
2.44	TISSS SOW Task (4.3.3) - Phase I Interim Design Phase Presentation	46
2.45	TISSS SOW Task (4.3.4) - Phase II Two Critical Milestone Reviews (CMR)	46
2.46	TISSS SOW Task (4.3.5) - Phase II Quarterly Interim Progress Reports	47
2.47	TISSS SOW Task (4.3.6) - Phase II Two Open Industry Reviews	47
2.48	TISSS SOW Task (4.3.7) - Phase II FCA And PCA At RADC	49
2.49	TISSS SOW Task (4.3.8) - Phase II Final Presentation	49
2.50	TISSS SOW Task (4.3.9) - ECLIPSE Kick-off Meeting	49
2.51	TISSS SOW Task (4.3.10) - ECLIPSE Technical Interchange Meeting	49
2.52	TISSS SOW Task (4.3.11) - LRM Standards Meeting Participation	50
2.53	TISSS SOW Task (4.3.12) - ECLIPSE System Requirements Review	50
2.54	TISSS SOW Task (4.3.13) - ECLIPSE Final Government Meeting	50
2.55	TISSS SOW Task (4.4) - VHDL/TVL Validation Software (Option)	50

3	TISSS MASTER SCHEDULE	50
4	TECHNOLOGICAL INNOVATIONS	52
4.1	Large Ada Program With DOD-STD-SDS Methodology	52
4.1.1	Standards Outlining The Ada And DOD-STD-SDS Process	52
4.1.2	Applying The Ada Structure To DOD-STD-SDS . . .	52
4.2	Hybrid Data Base	53
4.2.1	The Requirements For A Hybrid Database	54
4.2.2	Design Issues For The TISSS Administration Subsystem	55
4.2.3	Data Integrity	56
4.2.4	Data Protection	57
4.2.5	Transaction Logging	57
4.2.6	Locking	58
4.3	Product And Test Specification Language	58
4.4	Automated Postprocessing	60
5	TISSS LESSONS LEARNED	62
5.1	COMMON CODE	62
5.2	COMMUNICATIONS	63
5.3	CONFIGURATION MANAGEMENT	64
5.4	DID TAILORING	66
5.5	DOCUMENTATION	69
5.6	LINEs OF CODE (LOC)	71
5.7	METHODS	72
5.8	METRICS	75
5.9	PERSONNEL	75
5.10	PROTOTYPING	77
5.11	SCHEDULES	77
5.12	TOOLS	78
6	STATISTICS ON COMPLETED TISSS PRODUCT	80
6.1	Human Effort	80
6.2	Documentation Page Count	81
6.3	Deliverable Source Code	85
6.4	Test Code	88
6.5	Commenting	88
7	PUBLICATIONS PRODUCED	91
8	GOVERNMENT PATENT	91
9	TISSS INSERTION RECOMMENDATIONS	92
9.1	Upgrade To Current Versions Of Embedded Software	92
9.2	Upgrade To IEEE 1076	92
9.3	Upgrade TVL	92
9.4	Additional Test Philosophies	92
9.5	Beta Sites	92
9.6	Data Availability	93
9.7	Postprocessors	93
9.8	DLA/SPO Coordination	93
9.9	Insertion Issues From The TISSS Industry Review	93
9.9.1	How To Achieve Effective TISSS Buy-in By Merchant IC Houses	94
9.9.2	TISSS Usage In Plant	94
9.9.3	Standard Product Integrated Circuits (IC) . . .	94
9.9.4	Application Specific Integrated Circuits (ASIC)	98
9.9.5	Form, Fit, And Function And BIT Or BIST	98
9.9.6	A Special Case For ASICs	98
9.9.7	Where Is The Complete TIF ?	99

9.9.8	Update SUM Version Information For COTS	99
9.9.9	Update SUM Hardware Description Information For Laser Printers	99
9.10	TISSS Test Philosophy Minimum Test Set/Additional Tests	101
10	CONCLUSIONS AND RECOMMENDATIONS - TISSS FUTURE DIRECTION	102
10.1	Modeling - Hardware Accelerator	102
10.2	Test Specification (TDL) And Description (TVL)	103
10.2.1	Test Description Language (TDL)	103
10.2.1.1	Language Standardization	103
10.2.1.2	Additional Test Macro Types	104
10.2.1.3	Additional Semantic Rules	104
10.2.1.4	Object Names	105
10.2.1.5	Audit Exceptions	105
10.2.2	Detail Specification Requirement/Build	105
10.2.3	Test Vector Language (TVL)	106
10.3	Component And Board Level	106
10.4	Database And Tool Extensions	107

APPENDIX A FINAL TISSS STATEMENT OF WORK

APPENDIX B TISSS FOUR COLOR BROCHURE

APPENDIX C SELECTION OF COMPUTER SOFTWARE CONFIGURATION ITEMS
 (CSCIS)

APPENDIX D PATENT APPLICATION

D.1	POSTPROCESSOR OPERATION - INTRODUCTION	D-1
D.2	POSTPROCESSOR OPERATION - THEORY	D-1
D.3	POSTPROCESSOR OPERATION - IMPLEMENTATION	D-2
D.4	POSTPROCESSOR OPERATION - EXTENSIONS	D-3

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



1 INTRODUCTION

When the Tester Independent Support Software System (TISSS) development was begun in the fall of 1985, TISSS was one of the first large Ada Software development programs undertaken by Harris Corporation and the first project to follow DOD-STD-SDS (early version of MIL-STD-2167) methodology using Ada. The program was completed on cost and on schedule in July, 1988, an accomplishment that appeared extremely challenging in 1985 when the program was awarded because of the requirement to develop new technology on a tight schedule with a limited budget. In fact, it was an outstanding accomplishment according to a recent presentation given by T. Capers Jones of Software Productivity Research, Inc., Cambridge, MA, who stated that of all software development projects in the U.S. over 64K Lines of Code (LOC), only one percent are completed on cost and on schedule. The rest are either cancelled outright or are completed over cost and behind schedule.

This document is the final technical report of the TISSS software development and is intended not only to describe the technical detail of the system design, but also to provide the reader insight into a large Ada software development project using MIL-STD-2167, emphasizing the various lessons learned from contractor experience. Remaining open issues for TISSS insertion into private industry and government use will also be discussed.

1.1 TISSS OBJECTIVE

The objective of this effort was to define, develop, and implement a system for the automated generation and maintenance of electrical test specifications and test programs for complex Very High Speed Integrated Circuits (VHSIC) and other Very Large Scale Integration (VLSI) devices. This was accomplished through the development of the TISSS which provides a capability for the Government to capture design data in a standardized electronic form and use this information to develop and maintain device and test specifications in a standardized, transportable, and computer-accessible format and to automatically generate Automatic Microcircuit Test Equipment (AMTE) test programs for lifecycle support of VHSIC and VLSI devices. This system supplements the current MIL-M-38510 approach that is based on manually-developed device electrical test specifications and test programs.

This effort consisted of two phases. Under Phase I, the design phase, Harris Corporation, Government Systems Sector, the TISSS contractor, studied various approaches, evaluated related software developments and defined a specification and recommended a design approach for the implementation of a TISSS.

In Phase II, the contractor implemented the TISSS. Specific tasks performed during this phase included:

- a. The preparation of TISSS System specifications and documentation to allow future preparation of interfaces to various Computer Aided Design (CAD) systems for the purpose of the derivation of the Tester Independent Data Base (TIDB) information. Specifications and documentation to allow translation of the TIDB into any test system language were also prepared.
- b. The development of the documentation of all software necessary to interface and derive the TIDB from one set of Government specified hierarchical CAD databases as well as all software to perform automated translation of the TIDB to a native language test program for one Government-specified AMTE test system.
- c. The development of the capability for the Government to perform validations of additional TISSS software of either the derivation or translation type.
- d. The development of a fully operational system which enables the Government to develop and maintain machine readable specifications (TIDBs) for MIL-M-38510 microcircuits up to and including VHSIC level of complexity.

1.2 BACKGROUND

The present system for specifying, qualifying, and testing high reliability military microelectronics was designed for small and medium scale integration microelectronics and works well for these devices. The system is built around Government standards and specifications such as MIL-M-38510 and MIL-STD-883, which together, fully describe the various test requirements that are performed on military-qualified Integrated Circuits (IC's). Unfortunately, this methodology is inadequate for VHSIC and VLSI devices since electrical testing is a major issue not fully covered by the current process. Many other aspects of MIL-M-38510/MIL-STD-883 that pertain to VHSIC are being adequately addressed, but the approach electrical test is a major challenge.

In addition to the difficulty associated with qualification and specification development, there is another major and growing problem - test generation time and its associated cost. As devices become more complex, test generation cost increases dramatically. This is of great concern to the DoD because the DoD must both directly and indirectly pay for test programs for devices used in defense systems. And since there is currently very limited test program transportability, the DoD often pays for several dozen (perhaps even hundreds) test program preparations for many IC's. With the great projected cost of VHSIC test development, such continued cost duplication would be unacceptable and must be avoided.

To address these concerns, the DoD proposed to integrate and

automate the test specification and test generation process. This approach takes advantage of the CAD tools used to design IC's and minimizes the cost of generating device specifications. It also aids in test program generation for many test systems and minimizes the test program transportability costs while maximizing test adequacy.

The cornerstone of this new system is the Tester Independent Data Base (TIDB), machine-readable, device electrical test specification information that supplements the present hardcopy MIL-M-38510 slash specification. As it is machine-readable, it is possible to automate the specification preparation process and to use the data to automatically generate AMTE test programs.

2 TECHNICAL ACCOMPLISHMENTS

The tasks required in the final TISSS Statement of Work, as amended, is included in Appendix A for reference. For each SOW task, the following information is discussed where applicable : description of all technical work accomplished; information gained in performance of the contract (lessons learned); pertinent observations; nature of problems; positive and negative results; design criteria established; procedures followed; and processes developed.

2.1 TISSS SOW Task (4.1.1) - Review "Pertinent Research Topics"

Each pertinent research topic item was evaluated for applicability to the TISSS. Key concepts reviewed included: transportability of AMTE test language, and fault simulator environments. This work led to an open architecture for fault simulation environments and to approach the TISSS tester and CAD independence through a Test Description Language (TDL) based on microcircuit oriented product specifications and test interface specifications to standard Mil-Std-883 test procedures.

It was also determined that Automatic Test Pattern Generation (ATPG) would not be the major emphasis of the TISSS; but rather, validation of fault coverage was determined to be the best feature for an open environment. ATPG is a generally difficult problem in a well defined and controlled design environment and thus would perform poorly in a general purpose microcircuit qualification environment.

2.2 TISSS SOW Task (4.1.2) - Investigate Existing Software Packages

A team of evaluators was formed from different groups throughout Harris. The available software called out in Appendix 2 to the SOW were examined along with a variety of other software packages. The initial evaluation time span was approximately four months. Starting the latter part of 1984 and moving into 1985, the evaluation consisted of Data Base Management Systems, test generators, test equipment, CAD systems, and simulators. A

list of vendors contacted and their products are listed below.

Evaluated SOW Software

o HITS	o PREP
o LASAR	o ATLAS
o CADAT	o ADA

The following is the list of additional vendors contacted via telephone or letter. The list is partitioned by subject and alphabetized by vendor. The subjects are: Simulation/Test Generation, CAD Systems, Test Equipment, and finally Data Base Management Systems (DBMS's).

Simulation/Test Generation

Contacted Vendors -----	Product Name -----	Type -----
CAE System	IDEAL	Switch/Gate Simulation
Calma	TEGAS TEXSIM	Gate/Func. Simulation Mixed-mode Simulation
CDC	LOGIS	Gate Simulation
Daisy	DLS	Mixed-mode Simulation
Fairchild	INCYTE PREP	Test Generation Test Generation
Gateway Design	AIDSSIM AIDSTG	Fault Simulation Test Gen/Fault Sim
GenRad	HILO-2 HITEST	Fault Simulation Test Generation
HHB Softron	CADAT	Mixed-mode Simulation
Mentor	IDEA	Mixed-mode Simulation
Metheus	MILO	Mixed-mode Simulation
Phoenix Data	LOGCAP	Switch/gate Simulation
Prime Comp.	THEMIS	Mixed-mode Simulation
Silvar-Lisco	BIMOS HELIX	Mixed-mode Simulation Mixed-mode Simulation
SimuTec	SILOS	Mixed-mode Simulation
Teradyne	LASAR	Gate/RAM/ROM Simulation

Valid Logic	SCALD	Mixed-mode Simulation
Zycad	LE1002	Mixed-mode (H/W Accl)

CAD Systems

Contacted Vendors -----	System Name -----
Applicon	BRAVO
CALMA	CALMA
CAE	CAE
Chromatics	VLSI Designer
Computervision	CADDS2/VLSI Designer M
DAISY	Megalogician
METHEUS	METHEUS
Scientific Calc	MEDS
SDA	Design Automation S/W
Valid Logic	SCALDSTART
VIA Systems	VIA

Test Equipment

Contacted Vendors -----	System Name -----
ANDO	DIC 8035B
Cybernetics Tech	VIKING 200
GenRad	GR16/GR18
TAKEDA	T3340
Tektronix	S3270

Data Base Management Systems

Contacted Vendors -----	System Name -----
ORACLE Corp.	Oracle

One major area of concern was simulation. The proper tools had to be chosen to support the demanding needs of VLSI components. Review of TISSS requirements in this area led to the following desirable features for a simulation environment: structural and functional modeling capability, multiple fault type testing, fast and efficient algorithmic approach to fault simulation and comprehensive fault simulation reporting. Other desirable features were memory and CPU time saving simulation techniques, support of various design for testability (DFT) techniques, knowledge-based test generation, and a friendly user environment.

Review of the available simulation environments led to closer examination of five environments. These were HITS, HILO-2, HITEST, AIDSSIM, and AIDSTG.

HITS, the Hierarchical Integrated Test Simulator, is a Navy simulator supported by the Navy Air Engineering Center in Lakehurst, New Jersey. HITS was geared toward board level simulation rather than VHSIC components. It fulfilled many of the requirements stated above but fault simulation and test pattern generation was very CPU intensive. One of the major drawbacks was no Design for Test (DFT) support. It is difficult to express any kind of scan design in HITS. HITS is a unit delay simulator that supports bidirectional I/O pins. At the time of the evaluation, HITS supported only TTL technology.

HILO-2 had more to offer in the area of simulation. It supported other technologies, multiple logic states and strengths, nominal delay true-value and fault simulation, and bidirectional I/O pins.

Features between HILO-2 and AIDSSIM were compatible except that AIDSSIM used a fast logic/concurrent fault simulation algorithm. HILO-2 used parallel fault simulation and in addition, provided limited ATPG capability whereas AIDSSIM did not. However, like HITS there was no support for DFT techniques.

Two simulators that were evaluated, AIDSTG and HITEST, stood out as being able to perform the majority of required functions for VHSIC components. AIDSTG's claim was that it was a design verification tool as well as an ATPG for any one of four scan design disciplines. Namely, random access scan, scan set, level sensitive scan design, and scan path. AIDSTG and AIDSSIM together made up the test generator and fault simulation package that appeared desirable for TISSS.

HITEST was the most inclusive system and seemed to meet all requirements for a simulation environment. The main feature of HITEST was the use of a user provided knowledge data base for describing scan designs. A suite of software packages provided knowledge based interactive test generation and simulation systems which could be connected to a variety of visually effective graphics displays. HITEST design was based on the earlier implementation of HILO-2. Unfortunately, HITEST never became a product in the time frame required by the TISSS program.

HITS and HILO-2 were ultimately selected as the simulation environments to be incorporated into TISSS. A number of factors drove this decision. HITS was favored because it was a Navy simulator and already used by a number of government agencies and industry. HITS executable is very inexpensive. The major expense for HITS is joining the User's Group, which at the time of evaluation, was approximately \$2K per year. HITS was an acceptable board level simulator and could provide adequate simulation. However, for LSI and VLSI components, HITS had many shortcomings.

HITEST was the ideal simulation environment for TISSS. Unfortunately, it was still an alpha release and not a product. HILO-2 was chosen with the hope that an easier transition could be made when HITEST did become a product. Two other factors influenced choosing HILO-2. These were: the price of HILO-2 was approximately \$100K less than that of AIDSSIM and AIDSTG together, and support for the product was more established with identified training courses. At Phase II Contract Award, the upgraded version of HILO was available so HILO-3 was selected to be integrated into the TISSS.

2.3 TISSS SOW Task (4.1.3) - Design TISSS Architecture

The preliminary set of TISSS documents were created as part of the system engineering activity necessary to complete this task. The analysis of pertinent research and existing software packages and test languages led to the early partitioning of the system design into Computer Software Configuration Items (CSCIs).

The preliminary documents generated during the TISSS Phase I design included the Operational Concepts Document, the System/Segment Specification, a report on the hardware support requirements, and the development of the preliminary Software Requirements Specifications for each CSCI.

The program plan for the TISSS Phase II implementation was prepared and submitted as part of the TISSS Phase II proposal. All embedded commercial off the shelf software (COTS) items were defined and bid into the TISSS Phase II Proposal. These included the Oracle DBMS, the Genrad Hilo-3 Simulator, the Palette graphics system, the DEC Terminal Data Management System (TDMS), the DEC Common Data Dictionary (CDD), the DEC Ada Compilation System (ACS), and the DEC Virtual Memory Operating System (VMS).

The VHSIC Hardware Description Language (VHDL) Support Environment (GFP), was also planned to be embedded in the TISSS. The VHDL Analyzer (Compiler) without the simulator was integrated into the TISSS to support VHDL version 7.2 syntax and semantic functions, as well as translation to fault simulators. The Navy Hierarchical Integrated Test Simulator (HITS) was also provided as GFP and was integrated into the TISSS to provide fault simulation capability along with the commercial fault simulator, HILO-3.

The basic TISSS system was defined during this period as shown by Figure 2.3. The initial thinking assumed that an intermediate tester language might be universally translatable and portable to a variety of Automatic Microcircuit Test Equipment (AMTE) testers. A Test Description Language (TDL) was described which emphasized the description of a test program in portable statements that included AMTE actions and flow control. Unfortunately, typical AMTE test procedures rely upon and are closely coupled to the hardware architecture of the AMTE. It was felt that the universal TDL had to reflect a standardized test procedure from a microcircuit view and that detailed AMTE procedural requirements should be reflected elsewhere. Instead, a software requirements specification was created for each standardized test procedure that was tester independent. Tester dependency is reflected in the design of the test procedures.

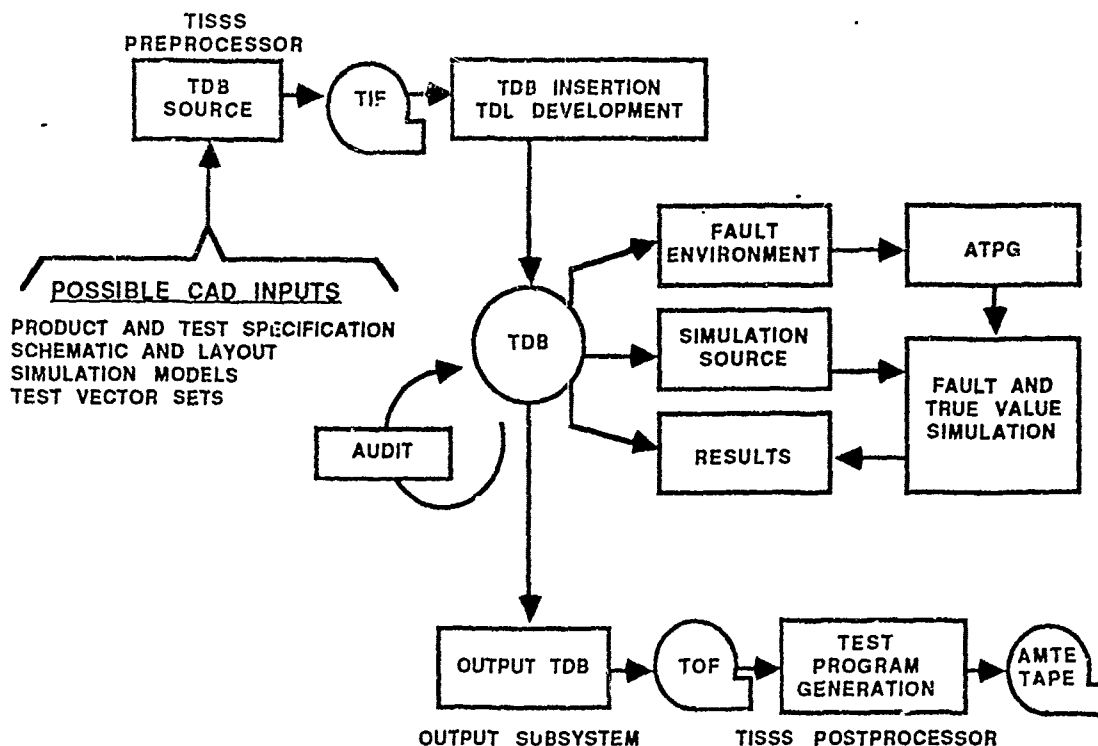


FIGURE 2.3 TISSS FUNCTIONAL FLOW

The proper atomicity of microcircuit test descriptions had to be determined. It was quickly concluded that a test (according to a MIL-STD-883 test method) within a MIL-M-38510 subgroup would be the proper atomic element of a test description between a microcircuit oriented test description and an AMTE oriented test procedure. Therefore, the TISSS system engineering team settled on a Test Description which appeared very close to a standardized Table III of the MIL-M-38510 detail specification. This put the burden of test description atomic element expansion to a detailed test procedure described in the native AMTE language into a TISSS AMTE postprocessor.

TISSS postprocessors can utilize highly tuned AMTE specific source code as input to the automatic test program generation process. A second major benefit of the postprocessor is for the long term logistics use of the TISSS Data Base (TDB). The TDB contains information which is independent of the current architecture of the existing AMTE industrial base. With DoD weapon system life cycles, typically two to four times longer than commercial AMTE life cycles, re-use of tester independent test descriptions leads to cost savings and higher quality when reprocurring microcircuit components.

The first TISSS postprocessor relied on a set of action directives embedded in GenRad GR-18 AMTE source code which direct macro expansion type actions by the postprocessor. The set of tester specific code with directives became input to the GR-18 AMTE postprocessor. The postprocessor symbolically processed the Test Description Language (TDL) and pulled from a specific Test Macro Library (TML) the procedure called out by the test description and then expanded that test macro procedure per the embedded directives. A patent application is expected to be filed by the USAF (RADC) for this concept, as shown in Appendix D. The end result is highly reliable, re-usable source code atoms for each test type according to the MIL-STD-883 test methods.

The next important area to define was the semantic requirements for a universally transportable and translatable test vector set based on deterministic input stimulus to a microcircuit and expected output response from a microcircuit. This format had to be translatable to simulators and AMTE, in keeping with the goal of having simulatable buses and test vectors within the TISSS.

It was decided that the use of a VHDL language basis for representation of these test vector sets would allow eventual expansion of the TISSS test vector set interface standards into the design world. This is very important because the complexity of these test vector sets demand a better human interface than a large file of hundreds of megabytes of ones, zeros and nanoseconds.

The TISSS data base presented a great challenge to the development of the TISSS. The experiences of other projects were evaluated in the use of relational database systems to support

electronic CAE, CAD and CAT data. These experiences strongly suggested that a detailed information management approach based on existing database technology would lead to failure of the TISSS program. Instead, new ground was charted with the development of the concept of a hybrid database using the important capabilities of the DBMS technology for metadata management, and existing hierarchical file systems to store language based representations of the CAE, CAD and CAT data. The metadata includes information about the electronic CAE and CAD data, but not the CAE, CAD and CAT data itself.

The TISSS metadata requirements were defined through an information model. The CAE, CAD and CAT data items went through formal syntax definition. The information model and the formal syntax and semantics definitions for each data structure provided a top to bottom definition of the information to be managed in the TISSS, and allowed each TISSS function the ability to access, read and modify the well defined database.

The creation of this hybrid, yet uniform, information model of the electronic CAE, CAD, and CAT data allowed long term re-use of this information, and extension of the TISSS. Further, a common set of parsing and analysis tools was defined for each language based descriptive item. These tools were then re-used for each interface to these descriptive items.

In every case, each syntax and semantics processing tool prepared an internal binary data structure which could be traversed directly. In addition, a large set of common object oriented procedural interfaces were defined. These interfaces allowed data requests which were independent of how the binary representation was stored. For example, get next node declarations from a netlisted VHDL structural model would be an object oriented procedural request of the VHDL model interface tools.

Each new TISSS tool will re-use these information model interfaces developed for the first TISSS release. Unfortunately, not all interface needs could be kept object oriented. So for example, when the TISSS is upgraded to IEEE 1076 VHDL, some small impact will be felt outside of the parsing and analyzing shared toolboxes. However, these impacts should be minimal.

A uniform user interface to all TISSS functions was defined very early in the TISSS system design. This user interface used mostly the same syntax and semantics from a user perspective. Most of the menu based interface works on models of real world objects. Functions performed map well to user oriented actions which might be performed in a CAE, CAD, or CAT environment in many smaller steps. However, the functions might be poor to use for iterative design. These functions assume the user has a verified design. These functions will catch data deliverable problems when the contractor has translated to these TISSS standard usage practices for VHDL or any other TISSS data base component.

2.4 TISSS SOW Task (4.1.4) - Evaluate Hardware And Software

Along with a variety of vendor software packages, two languages were evaluated for use in TISSS. The first of these was ATLAS, the Abbreviated Test Language for Avionic Systems. "Avionics" has been substituted with "All" in recognition of the wider application of the language. ATLAS is a language in which the unit under test (UUT) signal description is stated but not the instrument control needed to produce the signal. The concept behind ATLAS was appropriate for the test specification language required in TISSS. However, it was desirable for the TISSS test language to be in Ada syntax. This implied that a language parser could be developed that would be easier to implement and maintain. ATLAS had inherent complex syntactic and semantic rules as well as intricate intra/interstatement relationships. This complexity leads to error-prone coding processes and difficult language parser implementation. In addition, since the ATLAS language is primarily useful for analog rather than digital testing, it became apparent from the analysis that a standardized methodology to represent digital test vectors was needed.

Ada is the language of the future. The government required Ada for development of new software programs. Digital Equipment Corporation provided adequate Ada compilers. Harris had been using Ada PDL for years on software projects throughout the corporation. Review of existing tools and procedures lead to the acceptance of Ada as a workable language for TISSS.

2.5 TISSS SOW Task (4.1.5) - Evaluate Acceptability

User design data security was recognized early on as a prime requirement of the TISSS if the system was to receive broad user acceptance. An idea that came out of the Phase I study was to isolate the TISSS system computer in a secure area to protect user design information, but this concept seemed too cumbersome and expensive to be a practical solution. Manufacturers were surveyed to determine their willingness to provide proprietary data to the government for inclusion in the TISSS database. The responses indicated that an automated protection mechanism within the TISSS database would be required to prevent the accidental release of proprietary information.

The mechanism provided is the specification of TIF versus TOF outputs. A TIF (TISSS Input Format) contains all information known about a device, including proprietary information. A TOF (TISSS Output Format) contains only that data needed to generate a Detail Specification or an AMTE test program, none of which is proprietary. Further, the TISSS provides a security mechanism that authorizes a user the capability to generate a TIF separately from the capability to generate a TOF. Therefore, TOF generation, which is a common function, can be performed by clerks, while TIF generation, which is less common and requires more care, can be restricted to system managers.

The current security mechanisms within TISSS prevent only

accidental release of sensitive information. Two formal industry reviews were held to present TISSS to potential users for comment. At that time, industry representatives indicated additional resistance to releasing proprietary layout and schematic information to the Government. The concern was expressed that the information could be intentionally released to competitors, either legally or illegally, against the wishes of the original manufacturer. The Government's concern is that many systems must be supported for 20 years or more, while the devices used in the system may become obsolete and discontinued after only five years. Therefore, the Government requires enough information to allow the remanufacture of any device used within a system.

A consideration of these two positions prompted two suggestions. The first was that proprietary information desired by the Government be held in escrow, possibly at a location controlled by the manufacturer. The Government would verify the correctness and completeness of the information before accepting the data into the escrow account. Thereafter, the Government would have no access to the information without the knowledge and approval of the manufacturer, except under such conditions as specified in the escrow agreement. In general, the information would be released to the Government only if the device in question had been discontinued by the manufacturer but was still required for a government system.

The second suggestion was that layout information, which carries with it sensitive information about fabrication capability, is not really needed by the Government. The only reason to store layout data would be to remanufacture the device if it was ever discontinued by the original manufacturer. However, layout data typically strongly depends on a particular fabrication process, and the likelihood of an identical process being available several years later is small. Furthermore, most manufacturers have tools that will automatically produce a layout from schematics or gate level structural models. Therefore, there may be no need to store the layout information, and hence no need for the manufacturer to release the data.

There is one important use of layout data, however. That is the verification that the device actually manufactured conforms to the schematics and models released to the Government. This verification step can be performed by a government auditor, who typically has already signed a nondisclosure agreement with the manufacturer as part of the acceptance of a device specification.

2.6 TISSS SOW Task (4.1.6) - Perform A Phase I Preliminary Design

A preliminary design of the TISSS system was performed after the system requirements were defined through the Operational Concepts and the draft System Specification. (see SOW TASK 4.1.3)

2.7 TISSS SOW Task (4.1.7) - Conduct A Technical Presentation

The findings of the TISSS Phase I concept exploration were presented to RADC and Industry representatives on 27 February 1985 at Rome, New York. In this review, the TISSS Information Model and Architecture was reviewed in detail and a description given of how each Computer Software Configuration Item (CSCI) fit into the TISSS System Architecture, shown by Figure 2.7. The Modeling CSCI review included a description of the various behavioral and structural simulators planned for the system. Standardization issues were discussed where possible. VHDL and EDIF standards would be utilized by the TISSS to build on work already under way. Finally, the features of the TISSS itself were discussed from a users perspective and the potential cost and schedule benefits one could realize from the use of the system.

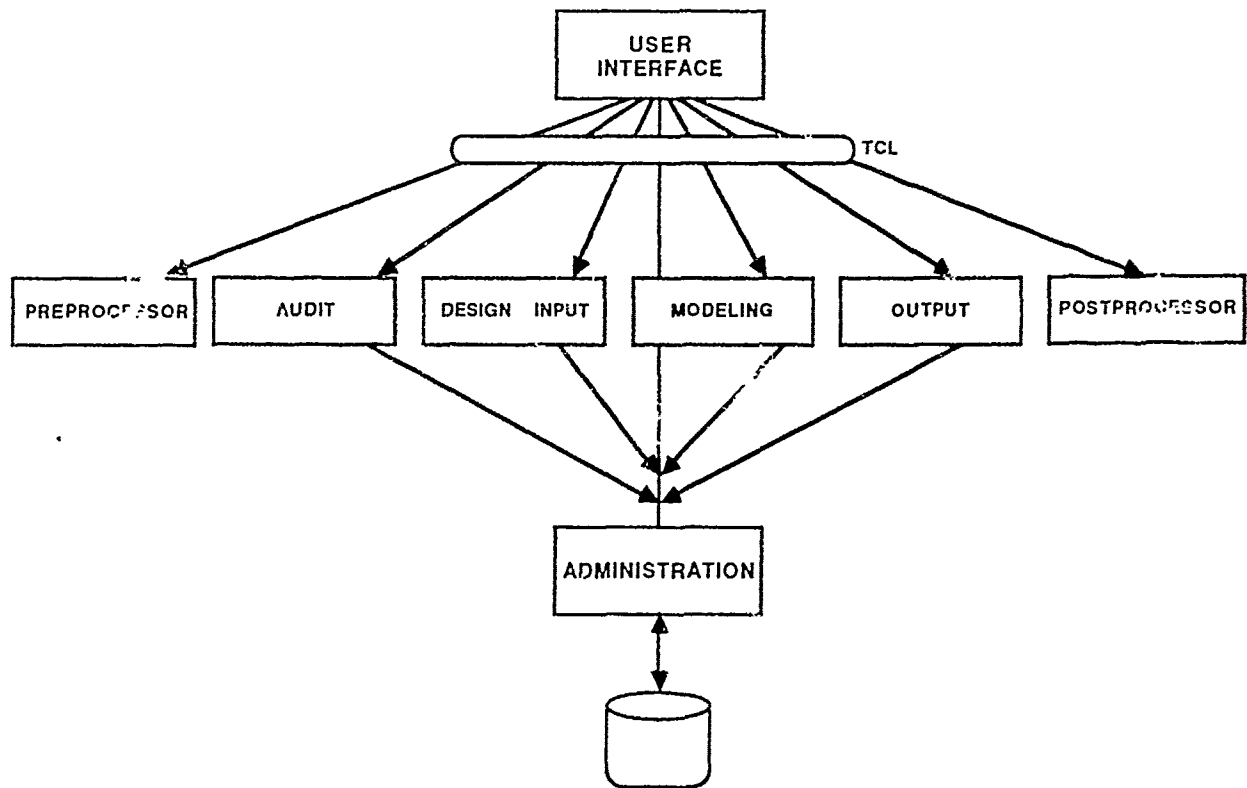


FIGURE 2.7 TISSS SYSTEM ARCHITECTURE

2.8 TISSS SOW Task (4.1.8) - Prepare A Technical And Cost Proposal

Detailed technical and cost proposals were prepared. The technical proposal emphasized the approach to each SOW task for Phase II and the risks that remained. The largest risk at the end of the Phase I effort was the availability of a stable GFP VHDL support environment. This problem was worked around in the proposal and the Phase II program.

2.9 TISSS SOW Task (4.2.1) - Perform A Phase II Preliminary Design

Upon contract award and before Preliminary Design could begin, it was necessary to rewrite and finalize the preliminary Software Requirements Specifications (SRSs) that had been developed during Phase I of the contract. This was due to further clarification and understanding of customer needs during the time period between the end of Phase I and the Phase II contract award, further developments in COTS software, transition from the Phase I to Phase II development team, and the use of requirements language and scoping terminology necessary for completion within a fixed price environment. Sufficient time was not budgeted for this effort which led to difficulty and additional cost in later phases.

One difficulty at this time was in performing requirements traceability from the System/Segment Specification (S/SS) and preliminary SRSs to the new SRSs. Many of the requirements in the original documents were implied in the supporting text but not enumerated and thus, not easily identified for tracing. A note for future specifications is to ensure that all actual requirements are put into lists which can be tracked and monitored. Tools should be used for requirements tracing. It was attempted unsuccessfully on TISSS to develop an internal tool for this purpose. An internal System Specification Review (SSR) was held at the end of November, 1985, a critical internal design review that the government should have been invited to. The schedule may have been slightly delayed but change decisions could have been made early at the most cost-effective time.

The TISSS Preliminary Design for Build 1 (the TISSS System less the Modeling Interface (MI) CSCI) was performed from early December 1985 through March of 1986. The requirements for three CSCIs (Postprocessor, Output and Modeling Subsystem) were somewhat vague at this point, causing confusion later on. There was some difficulty in learning how to use the Software Development Standard (DOD-STD-SDS) with Ada. SDS made use of older, traditional software methodologies and languages which did not easily lend themselves to modern software engineering practices and Ada features. An internal Software Standards and Procedures Manual (SSPM) was developed and variations to the Data Item Descriptions (DIDs) were internally agreed upon. It would have been better to have delivered revisions to the SSPM and Software Development Plan (SDP) that were produced during Phase I

at PDR, CDR and TRR. The SSPM is a guidebook for software developers and was updated, reviewed and clarified as necessary throughout the development process.

Software preliminary design was performed by using Yourdon functional decomposition techniques, as shown by Figure 2.9-1, to define the Top Level CSCs. A chart describing the symbol conventions is shown in Figure 2.9-2. For a complete description, refer to the TISSS Software Standards and Procedures Manual (SSPM). Ada Program Design Language (PDL) was used to define and document the interfaces between CSCs and the compiler was used to check the consistency of these interfaces. Object oriented design was not used at this stage except at the gross level to break out the features like the TDL tool box to access the objects, that is, Test Macro Skeletons, Test Plan Requirement, Test Plan Build, Pin Definitions and Detail Specification Build files.

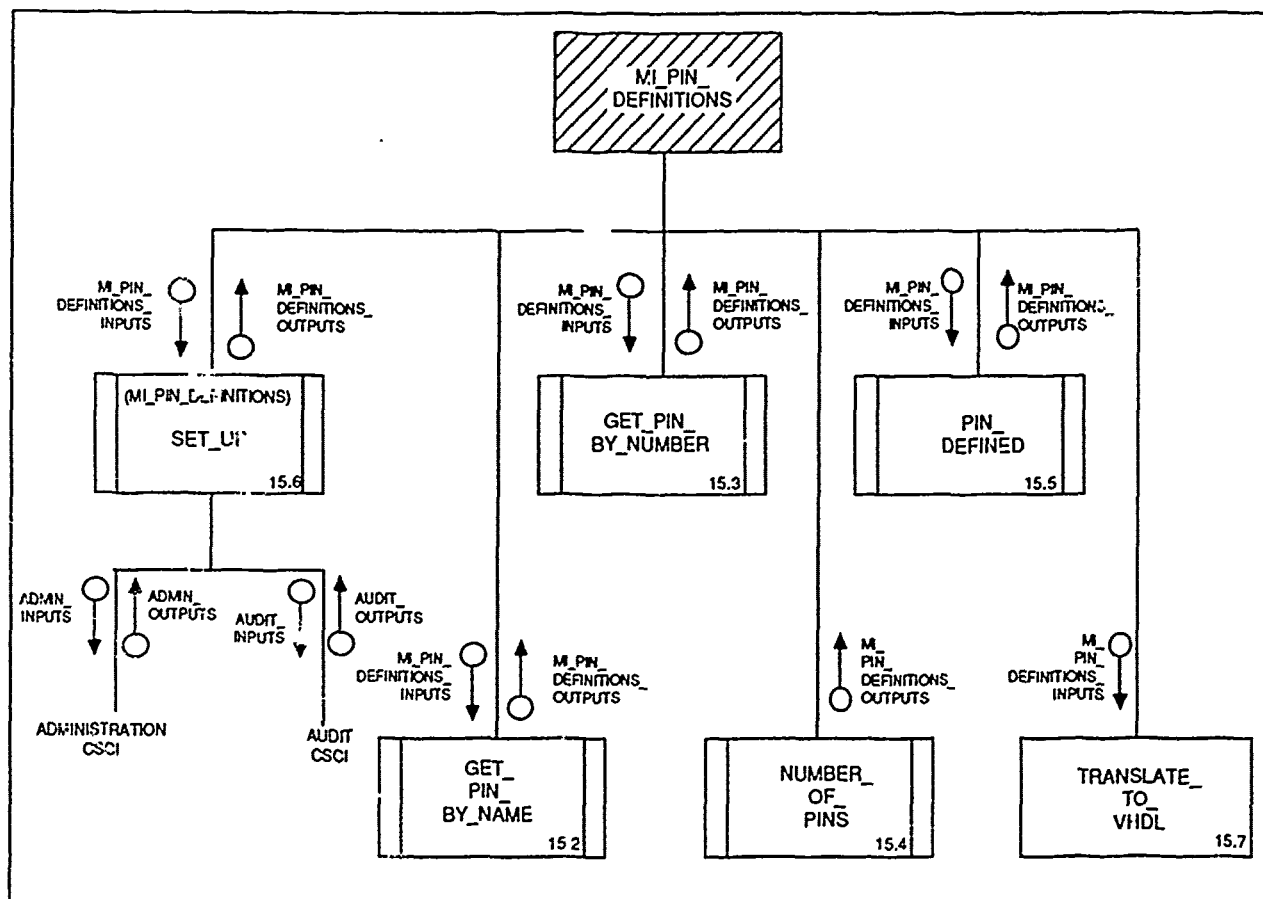


FIGURE 2.9-1 MI_PIN_DEFINITIONS TLSC

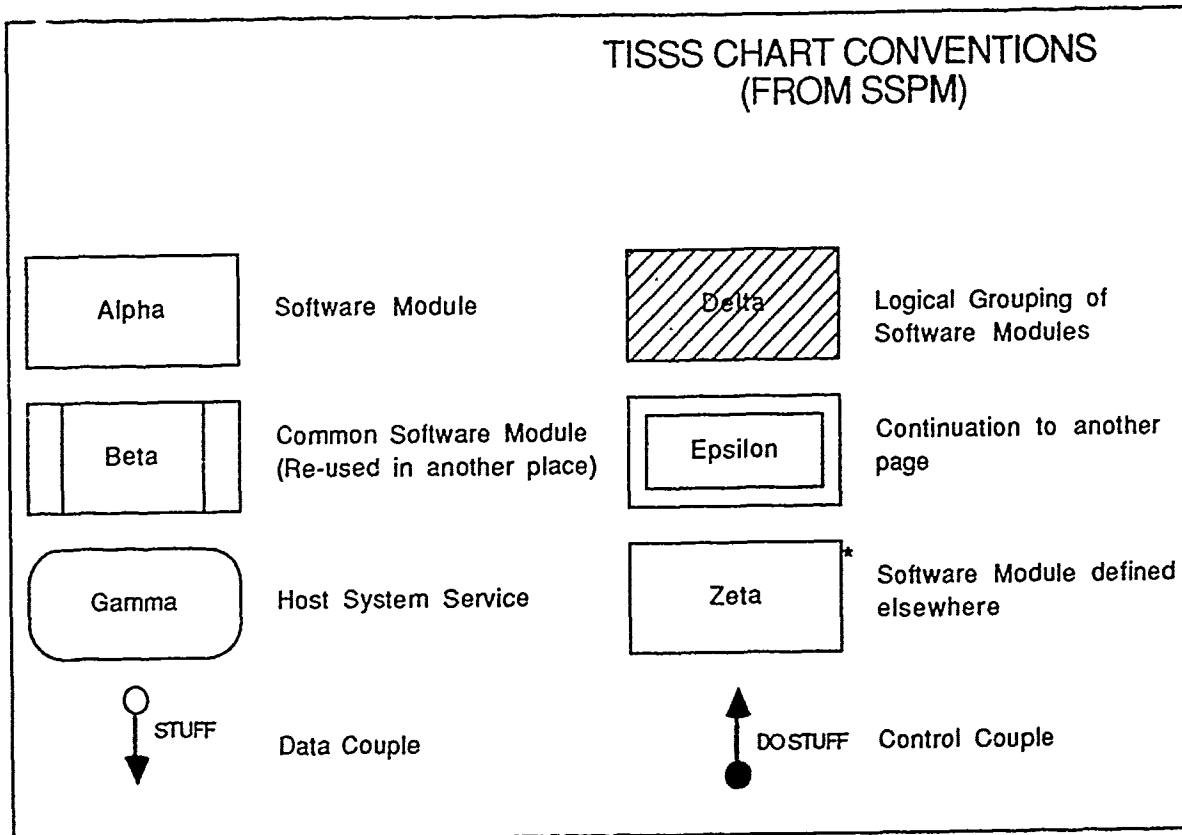


FIGURE 2.9-2 TISSS YOURDON SYMBOL CONVENTIONS

Prototyping was performed at this time in the following critical areas: user interface features and the use of the DEC COTS Terminal Data Management System (TDMS), Structured Query Language (SQL) interface to Oracle commercial data base, Ada tasking and the use of VAX communication services to prototype multiple users interfacing to a single database and recursive descent language parsing to verify the TDL grammar and approach for the TDL tool box. It was very advantageous at this time, to have the VAX 8600 target machine on-line and available to the TISSS software development team. It not only enabled the prototyping which was critical to the design process but allowed the compiler to be used to process Ada PDL; it allowed programmers to become familiar with the tools and environment; and it allowed configuration management of all documentation to be performed throughout the development process.

A difficulty that quickly became apparent was the CSCI levels that had been determined during Phase I and the required visibility to these CSCIs. First, there were too many CSCIs according to MIL-STD-483A criteria that should have been used for their establishment (see Appendix C for discussion of this issue). Because of these management boundaries and LOC estimates, the development of an optimal software solution with

common code was difficult. With the combination of more widespread use of object-oriented design and fewer CSCIs, more reusable packages would have emerged and less overall code would have had to be developed for TISSS.

Similar activities had to occur on Build 2 (the TISSS System including Modeling Interface (MI) CSCI of the TISSS contract. From late March 1987 to early May 1987, the MI SRS had to be rewritten by ECP direction. This time, a much better job was done in using requirements versus design terminology in creating precise, understandable, and testable requirements. All data formats expected and examples were given in the appendices. The time allotted for all of Build 2 was very short, the startup phase was particularly difficult because several activities from Build 1 were still in progress. An internal SSR was held prior to the official meeting. Again a few key government representatives should have been invited to attend for better understanding and to clarify direction before the design task was begun.

Preliminary design for Build 2 lasted from early May to the end of June 1987. This time, object-oriented design techniques were used which caused a more efficient overall design and created 8000 LOC of common code. The design team worked as a whole and assignments to TLCSCs were not made until later which allowed a broader understanding by team members and less duplication of effort. A new SSPM was developed at the start of Build 2 to incorporate lessons learned from Build 1. For example, a much more coherent mapping of Ada to MIL-STD-2167 was accomplished and stricter file naming standards were enforced. A tool was developed internally to produce the STLDD and SDDD documents from Ada PDL or code which was a great boost to productivity and morale. Prototyping again was a critical factor as all interfaces to COTS software were prototyped: interfaces to the HITS and HILO simulator, especially in determining how errors were reported to the user and how the software should process them and also prototyped was the interface to Intermetrics Intermediate VHDL Annotated Notation (IVAN) routines.

2.10 TISSS SOW Task (4.2.2) - Conduct A Detailed Design

Detailed design for TISSS Build 1 began immediately after PDR in March 1986 and continued until September 1986. The approach taken was to divide the design time into two stages and have an internal design review after the completion of the first stage. During the first stage, the top-level CSCs that were defined during Preliminary Design were further elaborated into lower level CSCs and units until all units were identified. Yourdon diagrams were used to illustrate this decomposition. Ada PDL was used to create a draft for the specifications of each lower level CSC and unit. The TISSS errors package was completed as far as knowledge allowed. At this point the design was reviewed internally by the chief programmer and cost account leaders.

In the second stage of Detailed Design, the PDL bodies were written for each unit using Ada PDL. They were compiled for correctness as time permitted. The level of PDL varied from CSCI to CSCI depending upon the understanding of the requirements and the time available. For example, in the Postprocessor CSCI, establishing requirements continued well into the design stage, the ratio of lines of PDL/lines of code was 50:1 in some areas. For other CSCIs, the ratio was as low as 2:1. As each TLCSC's PDL was completed, internal design reviews were scheduled with system engineering (SE), SQA, chief programmer (CP) and the cost account leader. Because of the large number of designers and TLCSCs - SE, SQA, and CP (which were each one person) could not attend all reviews or would have inadequate time to review the PDL. As the design evolved, the SRSS as written were not clear or accurate or some interpretation of the requirements was needed to fit the available resources. Although there were usually discussions and informal memos pertaining to requirements interpretation, these discussions should have been documented to reduce risk and to aid new people that came on the program. As a result, there are holes in the overall TISSS system requirements. SRSS should have been PTR'd, updated and maintained as the program went along. Again the system engineering staff was insufficient to support all of these desired tasks.

Prototyping continued to be important during Detailed Design. One large effort was to prototype the Design Input subsystem for feasibility. This allowed verification of concepts of interactively processing TDL, development of the Design Input menus, usage of advanced features of TDMS, and development of the overall Design Input architecture and common utility packages. The demo model of Design Input was shown to the customer for critical feedback and demonstrated at national conferences to illustrate TISSS features.

Some of the difficulties in developing the detailed design of TISSS centered around the documentation. One issue was how to incorporate the PDL into the SDDD. Since it was desirable to use the compiler to verify the PDL, the method chosen was to include the PDL files directly into the document and have the various sections of the document reference these PDL "listings". This eliminated some duplication of information in listing inputs, outputs and processing. One section that was written in the SDDD for each unit was a high level description of the function of the routine. This required that the designer update two files - the SDDD file and the PDL file - as changes were made to the design making it time consuming to keep them consistent. Also as PDL later evolved into code, these descriptions were included in the code header.

Another concern with the design documentation was the creation and upkeep of the Yourdon structure diagrams. For a person trying to learn or review the design of a CSCI, the Yourdon diagrams give a pictorial representation that is useful. The Yourdons were used in all of our internal and customer reviews. The tooling for the Yourdons was quite inadequate. The Yourdon

diagrams are drawn on a Macintosh with no tool used for checking consistency between drawings or levels. Several conventions were adopted in the internal SSPM to show such features as commonly used modules, modules that were expanded or continued on another page, and interfaces to operating system features. Since these conventions were developed in parallel with the design effort, there were some changes as various uses arose and some time was expended in rework. Also because of evolving standards, the resulting diagrams do not appear consistent between CSCIs. Since these diagrams were developed on a separate machine from the documentation text, no facility was available to merge the text and graphics. Each time the TISSS design documents were published, clerical help was used to cut and paste the figures onto the appropriate pages.

The Interface Design Document (IDD) was drafted during the first stage and completed during the second stage. The IDD shows all procedure calls across CSCI boundaries. The Data Base Design Document (DBDD) was also completed during the Detailed Design phase. Also, operating scenarios were developed for the Administration CSCI (AD) to give expected results when operating other CSCIs. Because AD provides data base access services to the other CSCIs, these scenarios provided information of the resulting actions that each data base access routine would effect. The scenarios are included in the AD SDDD, Section 6.2 (Notes).

The Detailed Design phase of TISSS Build 2 was short, lasting from July through September 1987. Again a two stage approach was used with an internal review occurring after the first stage. The first stage emphasis was elaboration of the design to the unit level where specifications and descriptions of each unit were developed. Yourdon diagrams illustrating the design to the unit level were completed. During the second stage the PDL bodies for each unit were developed. The goal of Build 2 was to write the PDL to as detailed a level as possible by CDR, in order to meet the short schedule for code, unit test and integration. The approach proved quite effective and design changes caused no major impact.

Most of the difficulties with design documentation that occurred in Build 1 were addressed and resolved in Build 2. The Design Document Generator (DDG) tool developed internally, generated the runoff file for the SDDD directly from the PDL or code files along with a template file and configuration file as inputs. This enabled the designers to perform all their work within one file, the .ada file, which was used first as a PDL file and then as a code file. Functional descriptions of the unit was placed in the code headers. The DDG tool placed the descriptions, inputs, output, processing, etc. into the appropriate sections of the document so no references to listings were required. Also the DDG calculated which units used other units and which data items were global and where they were used. During Build 1 this information for the SDDD was computed by hand which could lead to incompleteness and distrust for the accuracy of the data.

For Yourdon diagrams in Build 2, templates were drawn for each possible configuration between 1 and 7 boxes on a page. If more boxes were required, a drawing would be split into two pages. This enabled consistent drawings between various designers and eased the graphic creation process. The confusing convention of the '*' was dropped during Build 2. The problem of unavailability of a text/graphics merge tool still existed and the drawings were still pasted in by clerical help. A tool was developed at the completion of Build 2 to derive the Yourdon diagrams from the code through the DDG using the laser printer. This makes it easy to update the Yourdons with changes to the as-built implementation. This tool should be tested for usage during the TISSS Insertion and Maintenance Phases.

2.11 TISSS SOW Task (4.2.3) - Code And Test Each Software Unit

DOD-STD-SDS requires that Software Development Files (SDF) be maintained for each unit of a system. All SDFs must conform to established and consistent format. Each SDF file contains a copy of the code, unit test procedures and test results. A system with over 100,000 lines of code, such as TISSS, will consist of over 1000 units developed and tested. TISSS maintained all SDFs electronically to minimize the problem of storing the SDFs and to provide more thorough configuration and control.

The test procedures, test data and test results were stored as separate files. This prompted the creation of a tool to retrieve the electronically maintained SDF parts and combine them into the desired SDF format. The created SDF could then be printed or left in a specified on-line directory.

2.12 TISSS SOW Task (4.2.4) - Integrate And Test Software

DOD-STD-SDS specifies the use of top down testing. Some of TISSS was tested top down but in most cases bottom up testing was performed. Bottom up testing was required for CSCIs having complex data requirements at the top level and was best supported by testing lower level units first. Bottom up testing was also used when an LLCSC or a unit was required by many other LLCSCs and units.

Every unit in TISSS was tested using unit test procedures and reported using unit test reports as required by DOD-STD-SDS. Each developer was required to create a draft test procedure for each unit during the detailed design phase. Unit testing on TISSS provided failure detection and design correctness rather than requirements satisfaction. Requirements were tested during informal CSC and CSCI testing and again by the independent TISSS test team during formal CSCI testing

2.13 TISSS SOW Task (4.2.5) - Verify TISSS Operation

The technical work accomplished to verify TISSS operation consisted primarily of the TISSS Software Functions (Demonstration Scripts) which are included in the Software User's Manual (SUM), Volume 2, Appendix VI.2. In addition, activities during closure of the Build 1 phase of TISSS included tracking of interfaces to be tested. This activity was started by the Software Development organization and completed by the Formal Test organization.

The development of the demonstration scripts would have progressed more easily if their development had occurred after the revisions to the menu sections of the SUM instead of concurrently.

The focus of formal testing for TISSS should have been at the system level and not the CSCI level. This is an especially valid comment since TISSS CSCIs were usually not stand-alone processes. The TISSS demonstrations were focused on success, so recovery limitations were not necessarily or deliberately addressed beyond the CSCI level. The method used for demonstration script development largely ignored testing the interfaces between each of the CSCIs. However, this was recognized early in the script development and random testing of error conditions occurred whenever time allowed.

The design criteria that was established focused on demonstration of each menu functionality based on a success criteria. Development of demonstration scripts was crucial to determining TISSS capabilities as opposed to specific CSCI capabilities.

The formal CSCI tests followed the Software Test Plan. The scripts were written to serve as a TISSS tutorial as well as for System Validation. They did not map to the Contract Data Requirements List (CDRL). Twenty-Six (26) demonstration scripts were developed along with associated command procedures, ad hoc queries (*.SQL) and reports.

2.14 TISSS SOW Task (4.2.6) - Configuration Management Program

A configuration management (CM) program was developed to control the configuration baseline of the TISSS as described in the Software Configuration Management Plan. The plan identifies at a high-level the various procedures that were used to ensure proper administration of the Configuration Management tasks, namely, configuration identification, change control, status accounting, program trouble reporting, responsibilities of the Change Control Board (CCB) and auditing. For the actual operating procedures, methods and tools for TISSS CM, a much more detailed, streamlined and automated approach was necessary.

The CM library system evolved with the project and with the tasks necessary for each phase. The backbone of the CM library system is based upon the DEC VAX tool, Code Management System (CMS).

CMS is a library system for software development and maintenance. CMS stores source files in a library, keeps track of changes made to the files and records user access to the files. Initially TISSS CM started with one CMS library for storing the requirements documents at the completion of the requirements definition phase. Eventually the CM library system evolved into a hierarchy of directories and CMS libraries where each CSCI had separate CMS libraries for source, stubware, test, formal test set up, and SDDD documentation. The entire CM account currently maintains 70 CMS libraries. The CMS tool provides a good method of tracking different versions of files and the comment history facility allows changes to reference the appropriate Program Trouble Report (PTR) and Internal Software Delivery Form (ISDF) that caused a change. Using CMS simplified the tasks of keeping different baseline libraries as described in the SWCM Plan since all old versions were always accessible. CMS also provided features to manipulate combinations of files within a library with groups and classes. These will be used as TISSS is inserted into multiple sites and more than one version must be maintained.

In addition to the CMS libraries, another DEC VAX tool that was used quite heavily was the Ada Compilation System (ACS) product. ACS is the Ada program library management utility for VAX Ada. It provides an interface to the VAX Ada compiler and VAX/VMS Linker. VAX Ada programs are compiled and linked in the context of an Ada program library. Sublibraries are the primary mechanism for controlled sharing and concurrent program development in a multiperson project. The source code for TISSS was managed through a three-level ACS sublibrary system by CM. As developers made changes to the baseline TISSS, sublibraries of the CM libraries were created which inherited the context of the parent library and enabled quick testing.

Changes to the TISSS baseline documents and code were controlled by the CCB through approval of PTRs and ISDFs. Changes were only incorporated into the CM library with CCB approval. CCB meetings were held almost weekly throughout the duration of the program and the signature authority included the CCB chairman, Software System Engineering, Software Program Manager, Chief Programmer, Software Test and Integration, Software Quality Assurance and Software Configuration Management. There were a total of 944 PTRs and 1159 ISDFs written on the TISSS program and there were 108 CCB meetings conducted. The PTR and ISDF forms contain the basic information discussed in the SWCM plan. The forms were revised at least twice in the duration of the TISSS project to be tailored with more project specific and useful information. Initially the forms were printed on paper and filled in by hand. The latest version of the forms are kept electronically and edited by the person submitting the form. Hardcopies are retained by the CM clerk.

As the number of documents, files and libraries grew, it became necessary to develop automated command procedures for CM operations involved with creating and replacing versions in the CMS libraries and building, recompiling, and relinking the source

executables from ACS. At least 90 DEC Command Language (DCL) command files were written for these and other purposes. The command files provided a log of CM actions which were attached to the ISDFs and reviewed by the CCB. They were important for CM productivity, throughput and repeatability.

The Configuration Management personnel were organized to report to the Software System Engineer. The CM task was budgeted to support one CM clerk full time. It was found that this level alone was insufficient for a large Ada project. Engineering expertise was required in order to establish the CM library structure, establish the ACS library structure and coordinate the initial compile and link command procedures as well as develop other command procedures to automate the CM tasks. Many of the compilation changes and rebuilding of the CM ACS libraries were so lengthy that they were best performed at night. This approach freed up system resources during the day and prevented engineers from working with changing or obsolete libraries. The CM engineer should be one who is able to work late hours or who has access to a terminal from home because often these command procedures would need to be checked, corrected, and restarted at night to assure successful completion.

Difficulties in the CM area seemed to be related to lack of knowledge by the TISSS developers in CM procedure and proper completion of PTRs and ISDFs. A more comprehensively written CM plan would have aided in this understanding. Again this plan should have been updated as the procedures evolved. Insufficient information was provided on a PTR or ISDF form in some cases for a complete description of the problem or solution. Sometimes only portions of PTRs were closed with an ISDF. This made it very difficult in tracing PTRs to ISDFs and for cross referencing in the CM filing system. A better method would be to close the PTR and write a new one with a similar but reduced scope. One ISDF was sometimes used to deliver several files and close several PTRs. While this is a fast method of closing PTRs, it became difficult later on to trace the effects of individual PTRs. The number of PTRs closed with a single ISDF should be kept to a small number. The entire CM PTR and ISDF system require automation as much as possible with a data base centered tool to make CCB agendas, PTR and ISDF status reports, and PTR and ISDF creation fast and efficient.

2.15 TISSS SOW Task (4.2.7) - Software Quality Assurance Program

A quality program was developed to monitor the design, coding, test and integration of the TISSS as described in the Software Quality Assurance Plan. This plan was prepared to closely follow the standard Harris Quality Assurance processes and procedures approved for software development. Key features of the plan were to assure that the development team adhered to the Software Standards and Procedures Manual (SSPM); to assure that the test process guaranteed that all software requirements were met by each Computer Software Configuration Item (CSCI); and to assure that all contract deliverables met the specific Data Item

Description (DID) format and content requirements for each Contract Data Requirements List (CDRL) item.

The organizational structure implemented to assure that the TISSS development met the requirements of the functional and allocated baseline was to have a full time quality engineer assigned to the program, reporting to the Program Manager. In this way, any conflicts between the development organization and the baseline could be quickly resolved and any deficiency remedied. This concept of co-location of the developmental engineers and the quality engineer added a measure of efficiency because it allowed a timely review of the design documentation since the quality engineer was intimately involved with the development process.

The procedures followed by the quality organization as defined by the Software Quality Plan were to review and approve the Top Level Design and Detailed Design Documents for adherence to the Software Requirements Baseline as well as to monitor and approve the formal testing of each CSCI during the Functional Configuration Audit (FCA). The quality engineer sat on the Change Control Board and had approval authority for implementing any changes to the system and the documentation describing the system. For externally developed software products, the quality engineer audited the design documentation and the formal testing of the integrated products.

Throughout the TISSS development cycle, quality engineer staffing fluctuated considerably. Also during stages such as Functional Configuration Audit, at least four additional quality engineers were required to complete formal CSCI tests due to the number of CSCIs and lines of code within the TISSS program. These additional quality engineers allowed formal testing to complete within a reasonable although long period of time, but necessitated some unexpected changes since the documentation was interpreted differently at this final testing stage. While keeping overall costs lower with minimal staffing, formal testing could have been completed sooner if the additional quality engineers had been more familiar with the test procedures.

2.16 TISSS SOW Task (4.2.8) - Implement Planning And Control Procedures

Planning and Control procedures established for the software development were a periodic Line of Code assessment during the development cycle, monthly status and cost reporting, and control of the Program Plan or as referred to elsewhere, the Software Development Plan. Also, software testing control procedures and the use of a Program Support Library was established.

During each of the major software development activities in the Computer Software Development Cycle as shown by Figure 2.16-1, a projected Line of Code (LOC) estimate was established. The estimate after Phase I of the program was 58,855 LOC. A decision was made early on to use a comparison of the actual LOC to the

estimated LOC as a cost metric in this Design to Cost approach to software development. An internal Harris tool based on the Boehm methodology was used to count the LOC at each major activity as shown in Figure 2.16-2. Performance estimates were made and published in the design documentation as well as presented during the critical milestone reviews. The final LOC growth to 112,918 was partially based on a replan of the program caused by the delayed delivery of the VHSIC Hardware Description Language (VHDL) software. This eight week delay in the program resulted in a two build approach - Build 1 bringing everything except the modeling subsystem through to the integration and test phase, and Build 2, adding the replanned modeling subsystem and some critical enhancements.

Cost and schedule control was implemented by using the Harris Project Control System (PCS). The PCS system developed by Harris complies with the requirements of DoD 7002.2 and the C/SCSC Joint Implementation Guide, and provides information for preparation of the Cost Performance Report (CPR), Cost/Schedule Status Report (C/SSR), and the contract funds Status Report (CFSR). A top-down overview of the major management system elements which when integrated with PCS, provides a valid and logical sequence for organizing, planning and budgeting, accounting, analysis, and the timely incorporation of contract revisions, shown in Figure 2.16-3. This section describes and defines this generic management system utilized for program cost and schedule performance measurement.

PCS requires separation of the contract Statement of Work (SOW) tasks to a manageable and meaningful level. The development of a Work Breakdown Structure (WBS) is facilitated by selecting and assigning tasks for each element of the WBS to performing personnel. The integration of program tasks within the Harris functional organization results in the matrix relationship called the Responsibility Assignment Matrix as shown by Figure 2.16-4. At the intersection of the responsible organization and the WBS, a management level is defined for accomplishing all aspects of program work. It is at this intersection that the Cost Account is defined. The Cost Account represents the total effort managed by the cost Account Leader (CAL). The WBS is expanded to cover all task areas defined by the SOW and assignments made to the responsible program team member.

	BUDGET ALLOCATION OF LOC TO FUNCTION LEVEL	VALIDATE ALLOCATION OF LOC TO FUNCTION LEVEL	BUDGET ALLOCATION OF LOC TO CSCS	BUDGET ALLOCATION OF LOC TO UNITS	VALIDATE ALLOCATION OF LOC TO UNITS							
	REQUIREMENTS ANALYSIS	PRELIMINARY DESIGN	DETAILED DESIGN	CODE & UNIT TEST	BUILD 1 INTEGRATION BUILD 2 DESIGN	CODE & TEST BUILD 2	FUNCTIONAL CONFIGURATION AUDIT					
DATE	SRR 11-85	PDR 1-86 3-86	CDR 9-86	BUILD 1 UNIT TESTED BUILD BASELINE 3-87	MI/PDR 6-87	TRR 3-88	FINAL 9-88					
USER INTERFACE	9-85	7,445	10,390+	11,258+	13,481+	14,559+	11,157					
PREPROCESSOR	8,510	1,050	800	800	739	740	740					
DESIGN INPUT	8850	7,676	12,605	13,600	22,652	23,461	23,638					
AUDIT	8,100	10,165	11,850	12,798	15,552	16,232	16,327					
OUTPUT	65	645	1,000	2,895	3,776	4,590	4,903					
POST- PROCESSOR	2,100	6,875	7,685	10,495	16,484	18,856	19,562					
ADMIN	17,000	17,800	19,170	18,196	20,294	20,359	18,791					
MODELING INTERFACE	11,250	9,575	9,575	9,575	13,795	17,368	17,800					
TOTAL	58,835	61,305	73,075	79,617	106,773	111,790	112,918					

+ Includes 3555 for TDMS menu LOC

FIGURE 2.16-2 TISSS LINES OF CODE AT MAJOR MILESTONES

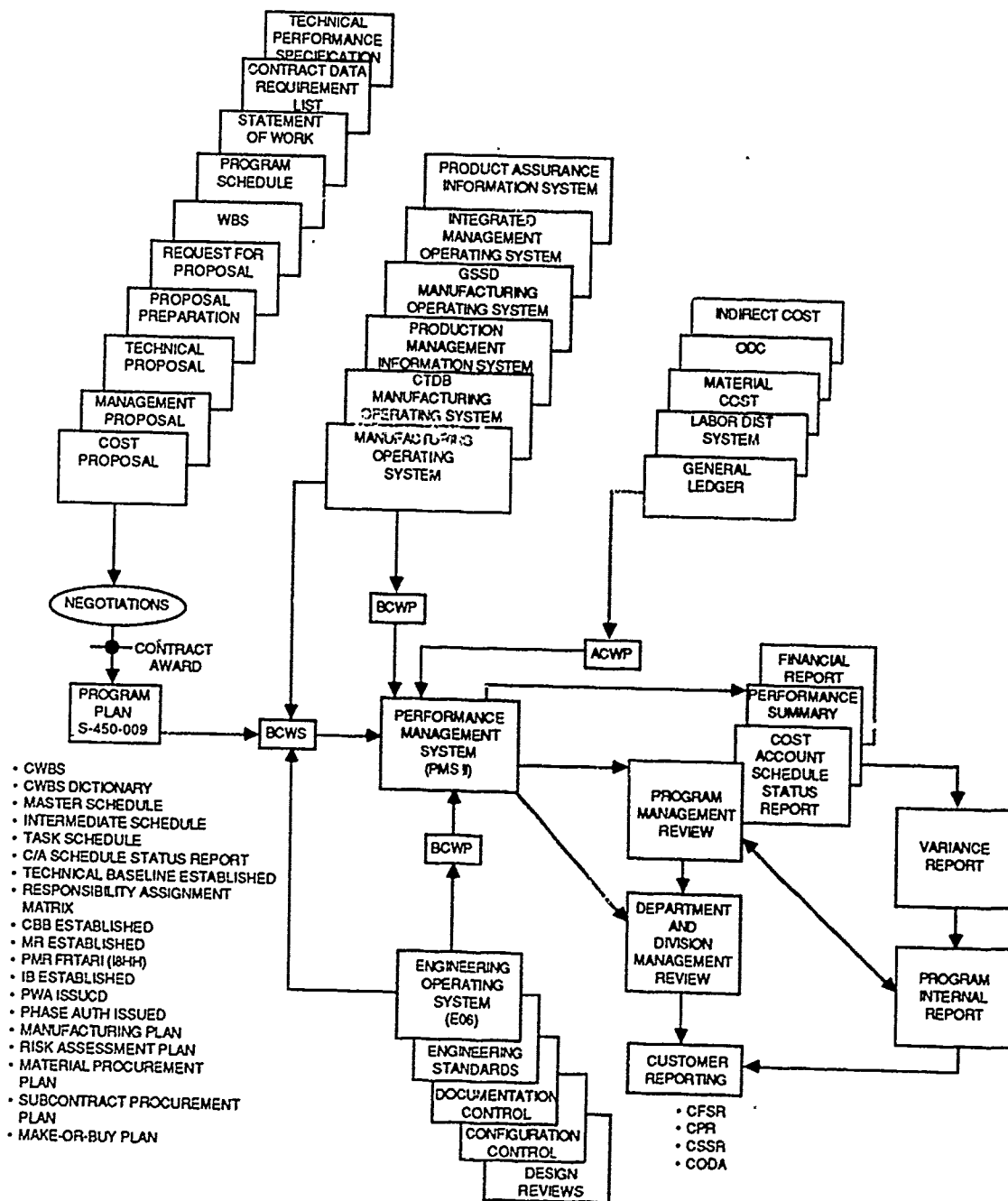


FIGURE 2.16-3 PROJECT CONTROL SYSTEM OVERVIEW

The Program Management Office defined the TISSS technical scope of work and developed a Program Master Schedule, a WBS Dictionary for each WBS element, and a Contract Data Requirements List (CDRL) schedule. These data along with the WBS define the program parameters and allowed the program baseline to be established. The Program Master Schedule, shown in Figure 3.0, is the initial step in baseline planning and it identifies the tasks to be performed, the sequence in which these tasks are to be accomplished, and the activity inter-relationships. Next, the WBS is developed where task elements were identified and assigned to the Cost Account Leaders who in turn established baseline budgets to perform the work. The WBS Dictionary provided a general and technical description of each task element in the WBS as it related to contractual work, which enabled the CAL to accurately budget the assigned work. The cost accounting system gathered actual costs by WBS element and reported them twice each month. The CDRL schedule was the data output required by the contract and was merely a listing of the data requirement schedule.

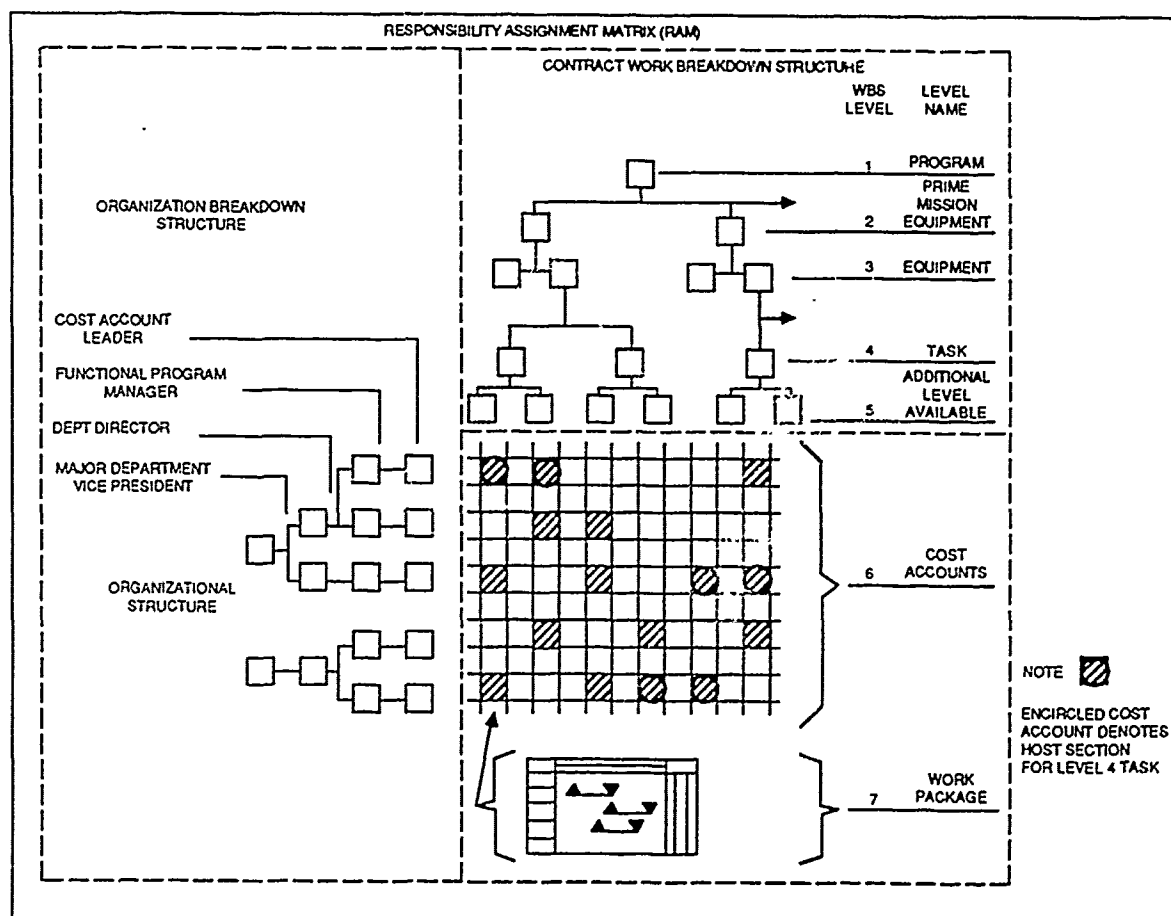


FIGURE 2.16-4 RESPONSIBILITY ASSIGNABILITY MATRIX

The Cost and Schedule Status Report (CSSR) system was used to monitor on a monthly basis, the Budgeted Cost of Work Scheduled (BCWS), the Budgeted Cost of Work Performed (BCWP) and the Actual Cost of Work Performed (ACWP) at the appropriate Work Breakdown Structure (WBS) level. These parameters were reported and any variance to plan above the allowed percentage explained. This status and cost reporting procedure gave not only the needed visibility to the customer into the program, but also, the program team was provided with trend data as well.

Program methodology for controlling policies and development procedures were documented in the Program Plan, the Configuration Management Plan, the Software Quality Plan and the Software Standards and Procedures Manual (SSPM). These documents specified the internal control procedures for the program and as directed by the contract, were under customer change control except for the SSPM which was delivered under Phase I and not a controlling document for Phase II. In addition, after government approval was granted for the Software Test Plan (STP), the Software Test Description (STD) and the Software Test Procedures (STPR), these documents were also placed under customer change control.

The program also went through the evolutionary development of a support library, where information from various sources was used to support the implementation of design features within the TISSS. Data such as the Computer-aided Acquisition and Logistics Support (CALS) standards and the MASA Support Requirements Document were invaluable in performing the ECLIPSE phase of the TISSS contract.

A dialog concerning TISSS field deployment has been conducted in the SUM with particular emphasis on minimum hardware and software requirements including power, layout, and environmental factors. User training and the change implementation process subsequent to deployment has also been covered. With the exception of the change implementation process after deployment, these topics are fully described in the TISSS Software User's Manual, the key reference material for a thorough understanding of the system.

2.17 TISSS SOW Task (4.2.9) - Coordinate TISSS With IDAS/VHDL Efforts

VHDL was successfully integrated into the TISSS. It is the interface specification for submitting behavioral and structural model descriptions. The VHDL environment, shown by Figure 2.17-1, consisting of the Analyzer and Design Library (DL) Manager, has been integrated into TISSS and provides a syntax verification. The output from the DL provides the intermediate format required for model translation to TISSS embedded fault simulators.

More explicitly, VHDL models were written for all TISSS packages, primitives, and representative test-bench elements. These VHDL test-bench components were tested for syntax by analyzing into

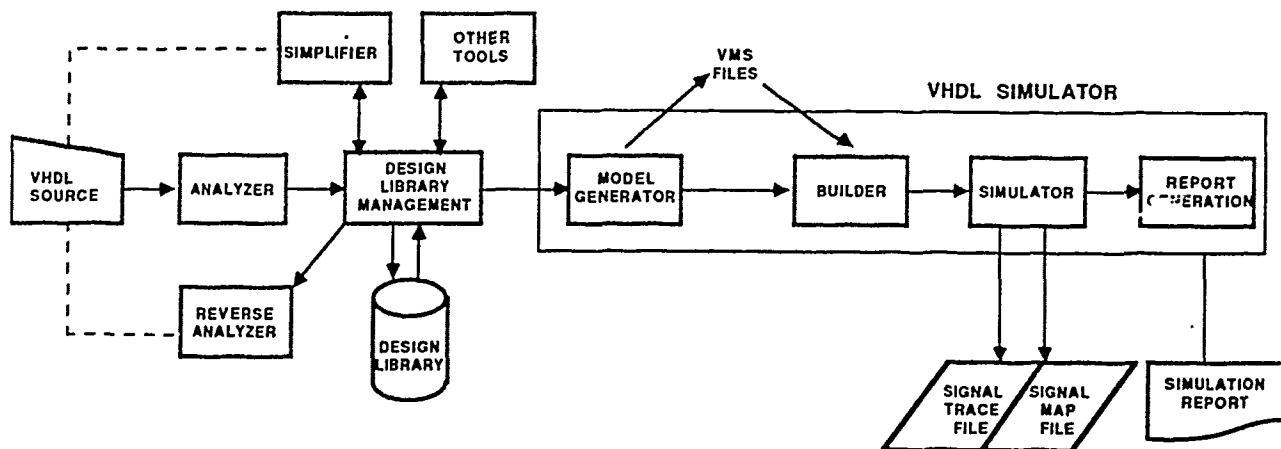


FIGURE 2.17-1 VHDL SUPPORT ENVIRONMENT

the Design Library (DL). The VHDL Test Bench Architecture is shown by Figure 2.17-2. Further testing was accomplished by model generating, building, simulating, report generating, and comparing the results against previous simulations. Specifications for the appropriate VHDL syntax of behavioral models, structural models, and test vectors were thoroughly documented and incorporated into the Interface Requirements Specification and Software User's Manual.

The Integrated Design Automation System (IDAS) effort begun with an assembled team from Hughes, Harris, Intermetrics, and GTE. Though the IDAS program was postponed/cancelled, it was apparent to Harris that TISSS itself encompassed some of the features that would be desirable for a VHSIC IDAS. The applicability of the TISSS to the objectives of the IDAS program is a strong reason to retain close ties between the two programs.

Harris has been actively involved in industry coordination efforts of TISSS standards. Explicitly, attendance at VHDL Analysis and Standardization Group (VASG) meetings, and active pursuit of an IEEE standard for the TISSS Test Vector Language (TVL).

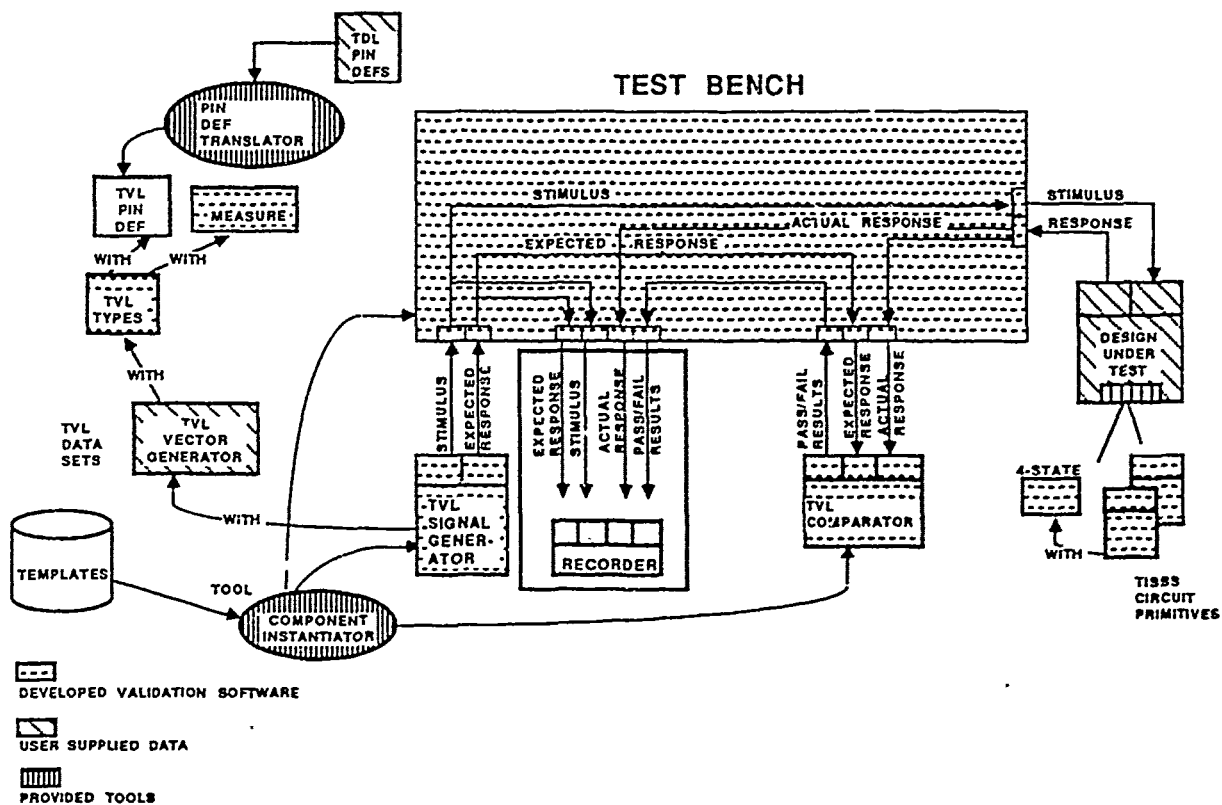


Figure 2.17-2 VHDL SIMULATION TEST BENCH ARCHITECTURE

The TVL effort was originally subcontracted out from Harris to Prospective Computer Analysts, Inc. (PCA). The goal was to expose industry and government agencies to the TVL. Harris provided a list of possible enhancements to be incorporated. Unfortunately, the task was not monitored as closely as was needed. TVL grew to such complexity that the original objective and intent of the language was no longer visible. Originally, TVL was to meet three top level goals. These goals were to: a) be a subset of VHDL and conform fully to VHDL syntax and semantics; b) precise enough to serve as a standard; and c) be easily translatable to a variety of simulators and AMTEs.

The major concern for the newly proposed TVL was a lack of validation to determine if the language could meet the above criteria. The TVL effort was brought back under the control of Harris. The TVL currently incorporated into TISSS will form the baseline for any enhancements to the language.

Recently, TVL has been presented from a requirements point of view to the IEEE SCC-20 ATPG Subcommittee and to the IEEE DASS Test Standards Subcommittee. Harris is the co-chairperson to the DASS Test Standards Subcommittee.

Under a future planned effort, the TVL enhancement task will produce a completely defined syntax for a TVL that meets the above stated criteria. Verification will be performed to insure compliance. A working group called Waveform And Vector Exchange Specification (WAVES) will provide the technical leadership for the task. The objectives of this working group will be to bring TVL before the IEEE DASS to receive feedback from Industry and government, with the eventual goal of obtaining IEEE standardization.

2.18 TISSS SOW Task (4.2.10) - Incorporate Fault Simulators Into TISSS

The HITS and HILO-3 simulation environments were successfully integrated into TISSS. The original intent was to incorporate HITEST developed by GenRad. HITEST was the most desirable simulation environment because of it's ability to support scan designs. Unfortunately, HITEST did not become available to the market in the required time frame. HILO-3, also developed by GenRad, was chosen in the hope that a smoother upgrade could be accomplished when HITEST did become a supported product.

HITS and HILO-3 both offer a true value simulation capability. However, only interfaces to the fault simulation and fault simulation report functions were incorporated.

The major interfaces required for the simulators are the VHDL model, consisting of the Interface Declaration, an Architectural Body, and the test vector set in TVL format. It was necessary to develop software that translated VHDL and TVL to the native simulator's model and vector language. A secondary interface is the TDL Pin Definition File (PDF). The PDF contains vital information on individual pins as well as pin groupings. Many of the TISSS subsystems use either the Pin Definitions File or the VHDL interface declaration that is consistent with the Pin Definitions File. The translators within the TISSS Modeling Interface Subsystem use the Pin Definitions File and the consistent VHDL interface declaration to assure interoperability within TISSS.

In order to provide a proper VHDL interface for the simulation environment, many system engineering hours were required. A subset of VHDL had to be defined that software could parse and easily translate to either the HITS or HILO modeling language. The scheme was to develop a basic set of TISSS structural primitives. The interfaces of these primitives would be thoroughly defined, as well as the architectural bodies. Structural VHDL models entered into TISSS could contain only these primitives supported in the basic set.

The VHDL interface for the Device Under Test, DUT, also had to be defined for behavioral and structural model descriptions. This interface required analyzing, model generation, and simulation to ensure these specifications would be simulatable if a VHDL simulation environment was integrated into TISSS.

The VHDL model is analyzed into the Design Library (DL). This results in an intermediate form known as IVAN. Ada code was written to interface with IVAN procedures that would extract data required for translation to the proper fault simulator format.

TVL translation occurs from TVL source. However, a TVL toolbox was developed for TISSS that parses the TVL source code and formats it into an internal data structure. Many TISSS subsystems work from this format. The TVL toolbox defines the data format that is the interface for translating TVL to the proper fault simulator format.

Simulation scripts containing actual HITS and HILO-3 commands are incorporated into TISSS. These scripts reside in VMS files and are executed to perform fault simulation and fault simulation report generation. Once simulation is completed, interfaces to the simulation reports are required. Stimulus and response information is extracted in order to generate TVL. Required data to fully populate TVL is not available from a simulation report. Therefore, a TVL template is generated that can later be edited. TISSS fault dictionaries are also generated from fault simulation report. .

The interfaces to the fault simulation reports gave developers the most difficulty. There were no written specifications for the ASCII text files. As versions of the simulators changed, so did these simulation report formats. It was necessary to "freeze" the HITS and HILO-3 versions so software could stay constant. HILO-3 version is "1"; HITS is version 12. In the future it would be beneficial to have access to some internal format of the fault simulator that would be unaffected by version upgrades.

2.19 TISSS SOW Task (4.2.11) - TISSS Interface Manual For Other CAD And Other AMTE

The TISSS Interface Requirements Specification (IRS) and Software Users Manual (SUM) fully document the language standards used by TISSS. These languages are the input to and output from the TISSS database. All CAD Postprocessors must generate these languages and all AMTE Postprocessors must read these languages, which represent the only interface between the TISSS and the external world. Therefore, these documents provide all the information required to develop new CAD and AMTE Postprocessors.

The file format to input information into the TISSS is called the TISSS Input Format (TIF). TIF is comprised of the Test Description Language (TDL), the Test Vector Language (TVL), and the VHSIC Hardware Description Language (VHDL) as well as IGES and EDIF graphics data. (VHDL is an industry standard. However, TISSS places restrictions on its use for structural models, the restrictions of which are documented in the IRS and SUM.)

TDL is designed to capture test related information about a microcircuit in a form compatible with the MIL-M-38510 detail

specification. The language is not tightly bound to the 38510 slash sheet, however, and this flexibility is a strong point, allowing for future changes in the specification as well as customized test approaches that deviate from the document. The net effect is to reduce the risk to the program since the program does not depend on a single interpretation of a MIL standard, it reduces the risk of obsolescence, and increases the probability of user acceptance. Therefore, the flexibility in the language, although it might appear to increase cost because less functionality can be "built-in", actually decreases cost, both initially and over the life-cycle of the system.

A strong attempt was made to make TDL "Ada-like". The sole value of being "Ada-like" appears to be the probability of less resistance from users who might be familiar with Ada and its syntax. On the other hand, TDL may be more complex and less flexible than it might have been because of the (partial) adherence to the Ada rules.

TVL is a subset (or application of) VHDL 7.2. (TVL will be upgraded to IEEE 1076 as part of future TISSS enhancements.) Like TDL, TVL suffers somewhat from its adherence to an existing standard, the most commonly cited problem being that TVL requires excessive storage space. Unlike TDL, which is only "Ada-like", TVL is truly VHDL, and therefore requires no processing to be used with a VHDL simulator. In this case the benefits of using a standard language, even one not well suited for the application, exceed the disadvantages. Note that the upgrade of TVL to IEEE 1076 will eliminate almost all existing objections to the language, based on the greater capability of 1076 as compared to VHDL 7.2.

2.20 TISSS SOW Task (4.2.12) - Validate And Verify All Software

Technical work accomplished in accordance with the Software Test Plan (STP) included:

- a. STP 4 - Software Development Files (SDF) containing unit and CSC informal test cases
- b. STP 5.2, 5.8 - Eight CSCI Software Test Descriptions (STD), CSCI Software Test Procedures (STPR), and CSCI Software Test Reports (STR)
- c. STP 3.1 - Pseudochip 2 data (VALCIRC CMV)
- d. STP 4.4 - Demonstration Scripts (refer to TISSS SOW 4.2.5)
- e. STP 6.3 - Validation Test Suites for CAD Postprocessor, Simulator, and TISSS Postprocessor

The Software Test organization should have been involved in TISSS from the beginning of the software development effort. Due to the the limited time and staff, overtime was required to accomplish the task of generating the CSCI formal test descriptions and procedures for each of the CSCIs. There was also a large flux of people between the test team and the software development team. This resulted in some training required each time a new member joined the team and some

confusion as well. Only due to some very key people was this task accomplished so successfully.

Generic test documents should have been developed early within the program schedule so that tailoring and addition of specific details would be the only changes needed. Instead, refinement of an extremely high-level methodology had to occur at a time when the detailed implementation should have already started.

Use of the DEC VAX/Test Manager (DTM) software was explored to enable easier re-verification of repeated formal tests. A more detailed method for using this tool is needed to take maximum advantage of its usefulness. It could be used for many of the unit or CSC tests as well as the CSCI tests. Also, another DEC product, Coverage and Performance Analyzer (CPA), in conjunction with DTM might enable more efficient testing. An evaluation of these two products and development of a proposed methodology should be performed.

2.21 TISSS SOW Task (4.2.13) - Validate Adequacy And Accuracy Of Generated TIDB's And Test Programs For New Devices

This task was one of the more difficult tasks undertaken during the TISSS Phase II effort. The approach was taken that each data structure in the tester independent database would be either independently analyzed for syntax and static semantics or generated by TISSS tools. Some data structures fell into both categories, in particular, all TDL based data structures. The syntax and semantics analyzers for TDL were developed and a combination of the VHDL analyzer and a TVL quick analyzer developed for models and test vector sets. It was quickly learned that expressing test patterns in VHDL would have severe performance limitations without a quick analyzer tailored to a small subset of TVL.

It was also decided that all TISSS functions would make use of the internal data structures provided by the syntax and semantics checking tools. These tools were called the TDL toolbox, the TVL toolbox, and the VHDL Analyzer and Design Library Manager. The VHDL Analyzer and Design Library Manager were furnished by the Government as GFP. This set of tools could validate TISSS generated TIDB data sets or non-TISSS generated data sets submitted in TISSS Input Format (TIF).

The final validation of a microcircuit TIDB is the postprocessing a tester independent program (TDL and TVL components of TIDB) through a validated TISSS postprocessor adequate to perform the test specified by the TISSS Output Format (TOF) subset of the TIF.

The TISSS GR-18 postprocessor acts like a limited virtual test machine and can be used to analyze resource requirements for the TDL and associated TVL test specifications. However, validation must be performed on at least one AMTE with the actual microcircuit to assure that the TIDB correctly describes the

microcircuit.

2.22 TISSS SOW Task (4.2.14)- Validation Test Suites

Validation Test Suites (VTS) were identified in the Requirements Baseline to test each critical tool function for the CAD Preprocessor (CP) CSCI, the Modeling Interface (MI) CSCI, and the AMTE Postprocessor (AP) CSCI. Several designs were captured and simulated on the Daisy CAD System to create data for the VTS. Each VTS was documented by a published VTS manual and a reserved Government VTS manual. Each VTS tests each critical tool function as identified in the Software Test Plans by illustrating valid test cases, proper results, and pass criteria. Each VTS, except the CP VTS, also illustrates invalid test cases, improper results, and fail criteria.

Ada software tools were written to translate Daisy netlist descriptions to a hierarchical VHDL format, to convert hierarchical VHDL to flattened VHDL, and to translate Daisy simulation reports to TVL format. The translated Daisy data and other data were used to create initial TIF related data sets which were input to TISSS via the Design Input Subsystem. TIF and TOF tapes were created for use by the Validation Test Suites (VTS).

Ada software tools were written to translate TVL to Daisy stimulus format and to automate comparisons of TVL data sets. The CP VTS was executed; translating TIF tape data sets to the Daisy CAD environment, simulating in the Daisy CAD environment, and translating the Daisy CAD data sets to proper TIF formats which were then validated. The CP VTS is limited by the nonincorporation of the VHDL Simulator.

Legal and illegal test data was created and stored in the TISSS database. The MI VTS was executed; translating structural models and vectors to HILO formats, translating HILO fault simulator reports to TISSS test vector fault dictionary format, translating HILO true value simulator reports to TVL, and validating all translations.

Ada software tools were written to generate pin definitions, fixture configurations, boilerplate TDL, and TVL stress data. The AP VTS was executed; translating TOF tape data sets into a complete program for the GR-18, running the GR-18 program, and creating output data sets which were then validated.

No significant problems were encountered during the VTS generation. The procedure followed for each published VTS manual (CDRL-B034) and reserved Government VTS manual (CDRL-B035) was specified by DID authority: DI-M-3413, which called out DI-M-3413/H-114-1, format type b: Development Program Manuals, was implemented.

2.23 TISSS SOW Task (4.2.15) - Generate Test Specifications And Programs

This option was not elected after the TISSS Phase II replan caused by the late delivery of the VHDL support environment. However, the original TISSS Phase II activity did include a task for Honeywell to create TISSS descriptions and specifications of two Honeywell VHSIC Phase I chips. Honeywell developed some of the software tools to migrate from their CAD system to TIF and made a first cut at the MIL-M-38510 device specification before the Government issued a stop work. The stop work was issued because the late VHDL support environment was also needed by Honeywell as GFP to complete their tasks for the VHSIC Phase I TISSS demonstration.

2.24 TISSS SOW Task (4.2.16) - Training Course

The training material consisted of a training outline, training material (an operational TISSS, viewgraphs, and a machine to display the TISSS menus, commands and outputs) and a training film. The prepared material was presented with minor changes whenever it was decided by RADC and Harris personnel (seven people were directly involved with the development and presentation of the material) that additional simplification of the information was needed. The course was divided into the following topics with designated functional users associated with each topic. However, most attendees decided to attend all the topics.

- a. TISSS Function Overview
- b. Introduction to Software User's Manual
- c. Introduction to TISSS
- d. Local Eve Plus Editor
- e. Creating TIF in CAD Environment
- f. Creation of Test Description with TISSS
- g. Model/Simulate Circuits
- h. TOF Distribution
- i. Generate Test Program
- j. Creation of Test Philosophy Library with TISSS (SRUs and their creation, TDL and Technological Information)
- k. Test Program Execution on GF-18
- l. Audit Functions (Design Existence Check, Syntax Check, Total Fault Dictionary, SRU Validation, Device Certification)
- m. Query Data Base (Pre-defined, Ad Hoc)
- n. Generation of Output Products (TIF/TOF Tape, MIL-M-38510)
- o. Manipulate SRUs Review
- p. Add Users and Devices
- q. Release Locked SRU
- r. Archive and Restore Device/Versions,
- s. Advanced TCL
- t. Introduction to DBA Activities
- u. ORACLE Startup and Shutdown
- v. System Startup and Shutdown

- w. VMS Support Environment
- x. Database Description
- y. Additional DBA Activities

In addition to the previously listed topics, there were daily hands-on sessions for students to become familiar with the TISSS functions and the menu system needed to exercise those functions. A comment program was made available to all attendees by RADC provided enabled an easy way to document suggestions for improvement, or to identify confusing topics throughout the course. These comments were reviewed at the Final Review and a plan established to address the comments in future enhancements.

Development of the training course as well as the course itself was delayed due to availability of people most familiar with the TISSS related concepts and to not interfere with other conferences which TISSS attendees would normally attend. With regard to the development of the training course, it would have been better if an individual had been appointed earlier in the contract to oversee the development of the course material. This person required experience in teaching as well as development of a format style for the presentation material and taped lessons as well as the exercises to demonstrate competence.

Additional screening of attendees should have been made to ensure that attendees had at least rudimentary knowledge of VAX VMS. This elementary knowledge should have included directory structures, moving around within directories, logging in/out of the system, use of an editor such as Eve or EDT. In the future, attendees without this knowledge should arrive a day early and learn commands such as SET DEFAULT, DIRECTORY, EDT, LOGOFF, and CREATE/DIRECTORY. DEC has an excellent on-line help library or self-taught computer primer courses for this purpose. Additional pre-requisites are included within the training material (CDRL B019). Individuals requiring more extensive pre-requisites should have attended the courses identified as pre-requisites.

Because seven key TISSS personnel were available during the TISSS course, questions and problems encountered during the hands-on sessions resulted in nearly one-to-one instruction. This enabled a greater depth of learning to occur for each individual.

Also included within this course were the exercises relating to each topic. Although only a few of the exercises were complicated, they all required integration of lecture information within the environment of working TISSS menus and commands. This was probably one of the best features of the course, because it enabled an easy method for attendees to status which TISSS functions they had learned.

2.25 TISSS SOW Task (4.2.17) - Install And Demonstrate TISSS At RADC

Although it was not required as part of the contract or as a CDRL, Harris felt it was absolutely necessary with a system the size and complexity of TISSS that a Computer Software Installation Manual (CSIM) be written. TISSS also interfaces with several Commercial Off-the-Shelf Software (COTS) products and their relationships had to be described in the CSIM. It is recommended on any future projects similar to TISSS complexity that a CSIM be required.

The CSIM discusses pre-installation activities involved with the installation and customization of COTS software, the steps to be taken during the installation of the TISSS software, and post-installation activities such as establishing user accounts and setting quotas. The installation of the TISSS accounts and software were automated through use of a master command procedure. The command procedure assumes the TISSS Configuration Management account has been delivered on magnetic tape. The use of the installation command procedure greatly aided in assuring the same steps were taken during each installation. It is helpful to have as much of the installation procedure automated as possible to prevent human error.

At the completion of the formal test for Build 1, Harris did an "informal" installation at RADC in October of 1987. This exercised the first draft of the command procedure and enabled its completion. Because the VAX 8600 at Harris was being used for the development of Build 2, the procedure could not be tested in its entirety at Harris. This early installation also gave RADC personnel advanced usage of most of the subsystems of TISSS, so that they could gain familiarity with them and be able to provide some feedback.

The CSIM was updated at the completion of Build 2 for features that had been updated and additional features that had been added for Modeling. Again, Harris was unable to thoroughly test the command procedure, since the TISSS system at Harris was being used for TRR and FCA preparation and testing.

At the completion of the CSCI formal test sell-off during May and June of 1988, the TISSS system was delivered and successfully installed through a two part installation (duration of 1 week per part) by Harris personnel in the presence of the RADC customer and IV&V contractor personnel. The first installation identified some minor errors in the CSIM and installation procedure (as expected since the procedure had not been previously tested). These problems were corrected at that time and solutions were later incorporated formally into the installation procedures in the CSIM document.

The second installation served to install some additional security features incorporated into the TISSS after the first installation and also to verify the new installation procedures.

The second installation ran quite smoothly. The CSIM document was amended slightly during the first run at RADC; then used during the second installation to verify all the steps were correctly outlined.

Following the installation, TISSS functions were demonstrated first by a subset of the Demonstration Scripts (written for System Validation, see SOW Task 4.2.5) mutually agreed upon by RADC and Harris Program Management. All scripts were executed successfully shortly after the final installation by RADC and the IV&V people.

2.26 TISSS SOW Task (4.2.18) - Beta Site Program (Option)

This option was not selected so no work was performed on this task.

2.27 TISSS SOW Task (4.2.19) - Preliminary And Final Technical Brochure

A Preliminary Technical Brochure was prepared in parallel with the software development at a time when many of the TISSS features were unclear. The text was developed by technical writers using data from program systems engineering personnel and photos taken of various people and equipment to describe TISSS operation. This initial effort, written by non-engineering personnel, was weak in describing the true capabilities and limitations of the TISSS. As a result of the decision to replan the program, it was decided to update the brochure to the final program baseline. The brochure was redesigned as a team effort by Government and contractor personnel. The revised brochure was delivered to coincide with the second Industry Review.

The approach taken to produce the Final Four-Color Technical Brochure, shown as Appendix B, was to involve the program team and the customer in a number of informal reviews prior to finalizing the text and format. In this way resources were conserved and the material was prepared and reviewed by a team who understood the system. Some of the photos taken for the preliminary brochure were used; others were selected from the Harris photo library. The text is timely in that it refers to the latest government thinking of how the TISSS will be used to meet future DoD microcircuit requirements. However, this brochure was written to describe the TISSS of the future where many testers and fault simulators are supported. Today only one tester is supported along with the two previously described fault simulators.

2.28 TISSS SOW Task (4.2.20) - Coordinate With ERADCOM Contractor

The Army portability of test software contractor was Titan Systems, Los Angeles, California. Harris worked closely with Titan Systems to establish an application domain for their advanced portability concepts. It was mutually determined that a

generic TISSS postprocessor with customizing modules per AMTE was quite feasible. However, the effort to develop such a postprocessor never came within the work scope of the TISSS Phase II effort but will be addressed during follow-on enhancements.

2.29 TISSS SOW Task (4.2.21) - Provide IGES Capability

IGES capability was added to the TISSS by the integration of the Commercial-Off-The-Shelf software (COTS) package, Palette. Palette allows creation and editing of IGES graphics on workstations and on the VAX VMS environment with color graphic terminals. Palette was part of the TISSS Phase II proposal baseline; however, the IGES capability became available later during the Phase II program. TISSS provides IGES 3.0 compatibility for technical illustrations and engineering drawings with the Palette software. Much effort was required to get the COTS vendor to provide complete IGES capability. Electrical applications for microcircuits are provided by VHDL for the various models and EDIF for the manufacturing representations. The TISSS currently defines the schematic as an EDIF type entity. It would be quite reasonable to use IGES for this representation and in fact far more commercial tools are available for IGES drawings. IGES has an electrical/electronic applications package for schematics and annotations. These additional IGES entities may not be supported by Palette at this time. This area needs further evaluation during the upgrade of TISSS to the IEEE 1076 VHDL standard.

2.30 TISSS SOW Task (4.2.22) - Provide SQL Graph Software

SQL-GRAPHS is an ORACLE software product that is supported by and used in conjunction with the Oracle data base management system. The SQL-GRAPHS software was included for integration into the Air Force host computer at RADC during the Replan of the TISSS program to provide graphical data base report capability.

2.31 TISSS SOW Task (4.2.23) - Define ECLIPSE OCD And S/SS

The ECLIPSE Operational Concept Document (OCD) was delivered as part of the ECLIPSE study effort. The ECLIPSE OCD describes the broad Air Force and DoD needs that initiated the development of ECLIPSE, a description of primary and secondary missions and their relationships to Air Force and DoD needs, and a description of the environment in which ECLIPSE would function. Additionally, it contains detailed discussions of ECLIPSE functions or capabilities, computer system functions required to support the ECLIPSE functions, ECLIPSE operator and user interactions. The ECLIPSE OCD also contains a discussion of the engineering methodology used to define ECLIPSE, the ECLIPSE Enterprise Model, descriptions of the ECLIPSE developing, using and supporting Government agencies, and further data of interest on the new concept of preemptive maintenance, LRM types and configuration control, and data security issues. A glossary is also provided. The ECLIPSE S/SS is discussed under SOW Tasks 4.2.25 and 4.2.29.

2.32 TISSS SOW Task (4.2.24) - Define ECLIPSE Exchange Standards

Volume II of the ECLIPSE Concept Exploration Technical Report (CETR), delivered as part of the ECLIPSE study effort, addresses the format specifications applicable to product definitions and other data items exchanged between organizations involved in the ECLIPSE enterprise. The purpose of this volume of the CETR is to document that portion of the ECLIPSE study concerned with data exchange standards, and to provide a stand-alone report of the standards recommended for ECLIPSE use. This volume contains an overview of the techniques that were used to accomplish the objectives of the standards tasks of the ECLIPSE study, a set of independent evaluations of MIL-Standards, other government standards, and industry standards that were identified as potentially governing the requirements for ECLIPSE information standards or otherwise relevant to the ECLIPSE study. The evaluations, where appropriate, propose extensions and modifications to the standards, provide rationale for these modifications, and define the potential roles of the standards in ECLIPSE. In some cases, the evaluations also provide a list of LRM design, test, and supporting data items covered by the standard. These data item lists provided feedback to the development of the ECLIPSE information model, which is itself an ECLIPSE standard for the management of information objects and their relationships and is included in the document. Additionally, the CETR Volume II describes the ECLIPSE Interchange Format, which documents the mapping of information standards to ECLIPSE information objects and describes the overall structure of the ECLIPSE digital data interface. A section is also included which describes the ECLIPSE data sets necessary to generate and represent a test program set for an LRM. The CETR Volume II emphasizes product and test data for Line Replaceable Modules (LRMs) and LRM-like items such as LRUs, SRUs, and PCBs.

Due to the high level of interest and activity in this area, additional funds for travel and research would have been useful. There were several working groups and meetings on data standards that were not attended, where ECLIPSE developments could have influenced the proceedings and eventual outcome.

2.33 TISSS SOW Task (4.2.25) - Define ECLIPSE Functional Requirements

The ECLIPSE System/Segment Specification delivered as part of the ECLIPSE study effort specifies the functional, performance, and interface system level requirements for the ECLIPSE. ECLIPSE requirements are focused on supporting the avionics LRM. This does not, however, preclude the support of LRM-like items (Printed Circuit Boards (PCBs), Shop Replaceable Units (SRUs), etc.) by ECLIPSE. ECLIPSE functions were selected by modeling the all functions in the entire ECLIPSE environment (enterprise) in an ECLIPSE Functional Model, and then identifying and elaborating on, those functions that ECLIPSE automates or interfaces with. The ECLIPSE Functional Model and supporting

narrative are included in the ECLIPSE System/Segment Specification.

Because of funding and time constraints, a complete appreciation of the goals, objectives, and functions of the Logistics Management Systems (LMS) modernization program (REMIS, DMMIS, etc.) was not developed until late in the study. This caused a replication of the functions of these systems in the ECLIPSE draft preliminary system specification, and resulted in some criticism at the SRR. These functions were deleted from the final documents.

2.34 TISSS SOW Task (4.2.26) - VHSIC TISSS Evaluation

Volume III of the CETR, delivered as part of the ECLIPSE study effort, contains the evaluation of the VHSIC TISSS concepts with regard to the ECLIPSE operational concepts. The CETR Volume III includes discussions of how TISSS can become a part of the ECLIPSE environment by supporting LRM components, and, in what ways TISSS can provide the basis for the development of the ECLIPSE system. A summary of the evaluation findings is included.

2.35 TISSS SOW Task (4.2.27) - Evaluate Tools For ECLIPSE

Volume III of the CETR, delivered as part of the ECLIPSE study effort, contains an evaluation of existing tools for possible integration into the ECLIPSE. These tools include CAE/CAD tools, Automatic Test Pattern Generators, logic and fault simulators, and TPS Automation tools. Summaries of the evaluation findings are provided, as well as recommendations.

2.36 TISSS SOW Task (4.2.28) - Evaluate Host Environments

Volume III of the CETR, delivered as part of the ECLIPSE study effort, contains an evaluation of candidate ECLIPSE host environments based on the developed requirements. An ideal host is compared to available products. Hardware and Software requirements are addressed, as are system organization and communication requirements. Information concerning Optical Write-Once, Read-Many (WORM) disk technology is also provided.

2.37 TISSS SOW Task (4.2.29) - Specify ECLIPSE System Requirements

The ECLIPSE System Specification, delivered as part of the ECLIPSE study effort, establishes the preliminary requirements for the ECLIPSE System. It specifies the system level functional, performance, and interface requirements for the ECLIPSE. This document identifies all applicable documents, all internal/external interfaces, and any other information necessary to establish a complete set of system requirements for the ECLIPSE. Appendices contain the ECLIPSE functional model, ECLIPSE information model and the ECLIPSE data dictionary.

2.38 TISSS SOW Task (4.2.30) - ECLIPSE Concept Exploration Technical Report

Volume I of the CETR, delivered as part of the ECLIPSE study effort, documents the results of the trips and telephone conversations by ECLIPSE team members in relation to the ECLIPSE study. Telephone and trip reports are contained in separate sections, with each section being organized chronologically. The content of Volumes II and III of the CETR are discussed above. Descriptions of the ECLIPSE system functions and characteristics are documented in the OCD and SS. Assumptions, where appropriate, are documented throughout all the ECLIPSE documents.

2.39 TISSS SOW Task (4.2.31) - Coordinate With Other ECLIPSE Contractor

A coordinated research and requirements development process was made with the additional ECLIPSE effort contracted by the Government. Daily teleconferences were made and joint monthly review meetings were held. Joint fact-finding trips and presentations were made to the various involved Government agencies and office space was provided at the Harris facility to help prepare inputs to the documents. The ECLIPSE TIM presentation was produced and presented as a joint effort.

2.40 TISSS SOW Task (4.2.32) - ECLIPSE Insertion Plan

The ECLIPSE Insertion Plan, delivered as part of the ECLIPSE study effort, describes a top level concept for the insertion of the ECLIPSE system into the USAF Air Force Logistics Command (AFLC). It identifies the AFLC organizations and other Air Force organizations that interface to ECLIPSE. It also provides a preliminary Line of Code (LOC) estimate for ECLIPSE.

2.41 TISSS SOW Task (4.3) Reviews

Technical reviews were generally scheduled on a quarterly basis throughout the term of the program. Critical Milestone Review meetings such as Preliminary Design Review (PDR), Critical Design Review (CDR), and Test Readiness Review (TRR) were conducted in accordance with the procedures established by MIL-STD-1521B (USAF), "Technical Reviews and Audits for Systems, Equipment, and Computer Software" while open presentations given to industry and government representatives and regular quarterly reviews were presented in contractor format.

2.42 TISSS SOW Task (4.3.1) - Phase I Kick-off Meeting

A Phase I Kickoff meeting was held on 28-29 August 1984 at Melbourne, Florida for the purpose of bi-lateral familiarization of the TISSS team and tri-service government representatives and to discuss the technical issues and the TISSS software development approach. Valuable insight and direction was provided from the RADC reviewers to scope the program toward meeting Government needs.

2.43 TISSS SOW Task (4.3.2) - Phase I Preliminary Specification Review (PSR)

A Phase I Preliminary Specification Review was held on 26-27 February 1985 at Melbourne, Florida to review contractor progress and to provide government critique and technical direction as the software system design evolved. Again, the RADC reviewers were most helpful in driving the TISSS team to consider a flexible approach toward test philosophy. The concept of a test philosophy library was developed during this period based on Government reviewer feedback. Honeywell was also represented at this meeting and provided constructive feedback as well.

2.44 TISSS SOW Task (4.3.3) - Phase I Interim Design Phase Presentation

A Phase I Interim Design Phase meeting was conducted on 3 October 1984 at Ft. Monmouth, New Jersey to review the initial system design of the TISSS. The basic top level design was presented and although much work remained to convert this design information to a producible system specification, the Government accepted and approved the design requirements and approach. It was during this time that continuous and open communications channels provided the valuable feedback and direction necessary to enable the contractor to design a system that could ultimately succeed.

2.45 TISSS SOW Task (4.3.4) - Phase II Two Critical Milestone Reviews (CMR)

Two Critical Milestone Reviews were specified by the contract Preliminary Design Review and Critical Design Review. The Preliminary Design Review was conducted on 15-16 January 1986 at Harris Corporate Headquarters in Melbourne, Florida. The purpose of the review was to present the top level design of each of the Computer Software Configuration Items (CSCIs) as well as to discuss the Software Test Plan and the Configuration Management implementation plan. Quality Assurance procedures were discussed as well as important future enhancements to TISSS. A government tri-service management critique was held on both days to provide the contractor direction for feature improvements through response to action items. The Critical Design review was held on 23-25 September 1986 again at the Harris Corporate Headquarters in Melbourne, Florida. The objective of this review was to review the Detailed Design phase of the TISSS system including all CSCIs except for Modeling Interface. Test Engineering test macro development data was reviewed in detail as was CSCI test methodology. To give the reviewers an understanding of how the TISSS system operated at a systems level, a TISSS System Scenario was presented showing how all the design information was prepared and input into the TISSS Data Base (TDB). The interactive test engineering data input was also covered as well as operations and system controller functions with the DEC VAX 8600 computer. Again, the Government tri-service management committee critiqued the presentation and provided direction and

scope through a request for action item responses.

2.46 TISSS SOW Task (4.3.5) - Phase II Quarterly Interim Progress Reports

The program was generally structured to have Critical Milestone Reviews (CMR's) on as close to a quarterly basis as practical, where the time between major reviews exceeded the three month period, interim progress reviews were conducted. These meetings were held at tri-service locations in order to expose as diverse a government population as possible to the TISSS concepts and progress on the software development. These locations were at Crane, Indiana for the U.S. Navy, at Washington, D.C. (Ft. Monmouth) for the U.S. Army, and at Rome, N.Y. for the U.S. Air Force. A total of five such meetings were conducted.

2.47 TISSS SOW Task (4.3.6) - Phase II Two Open Industry Reviews

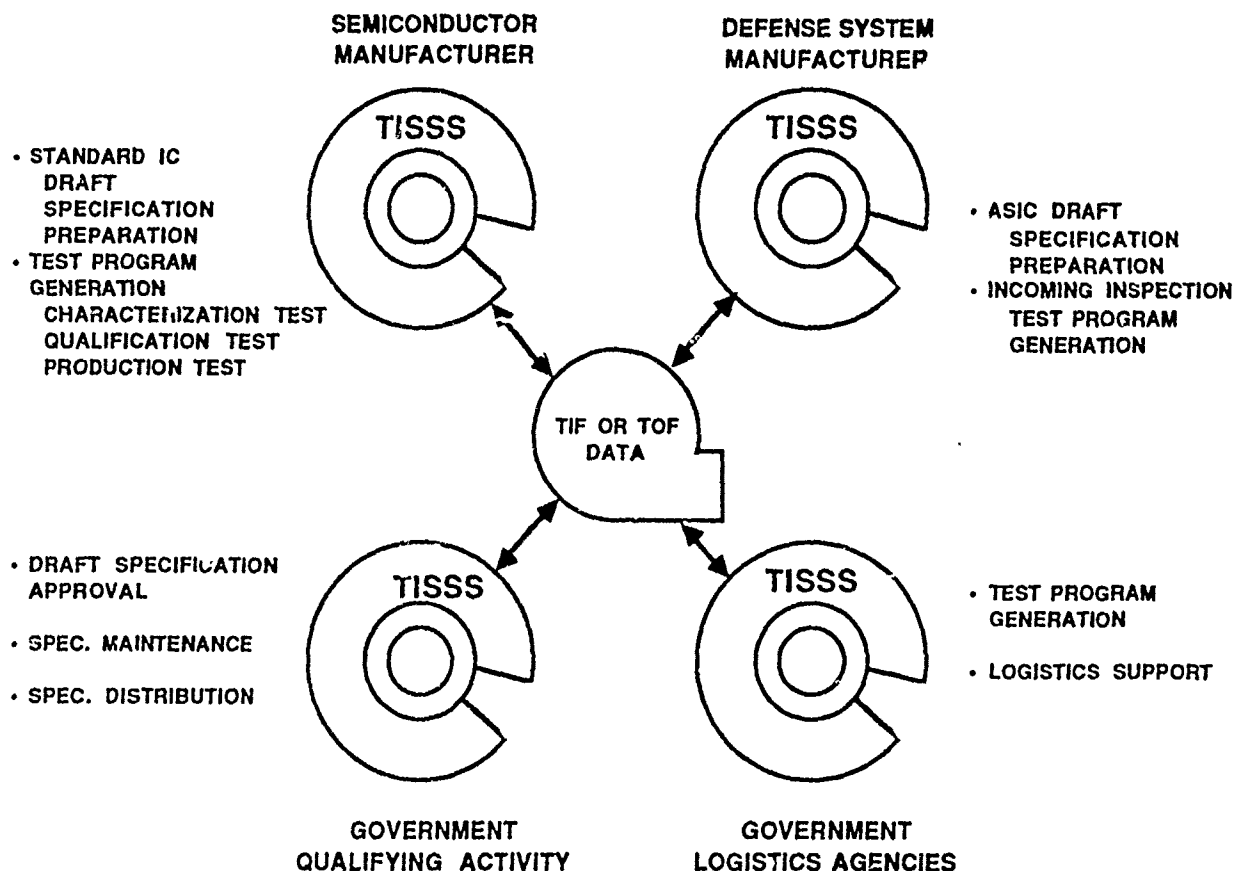
The first Industry Review for Phase II was held in Phoenix, Arizona October 15-16 1986. Approximately 50 representatives were present from government and industry.

The intent of this review was to present the concepts of TISSS Phase II. A technical overview was presented which emphasized TISSS objectives and top level features. Priority features were standard interfaces from CAD systems, the test specification language, and the tester independent data base. A software overview was also presented. At this point, Phase II Critical Design had been completed for all subsystems within TISSS with the exception of Modeling. Modeling had been stopped due to the development schedule of the VHDL environment. Focus for this review was on the software system architecture, software tasking, VMS configuration parameters, methodologies, and line of code estimates. Each subsystem presented reviewed functions and software architecture.

For the first time a TISSS user scenario was presented to Industry. The intent was to give reviewers a system level view of how the TISSS would interact with a user, as depicted by Figure 2.47. An actual on-line demo was given of the Design Input Subsystem. This exemplified the tester independence nature of TISSS by showing how a user enters test specifications for a particular device.

The Test Vector Language (TVL) developed for TISSS was introduced. The function of the language was presented along with the syntax and semantics.

The majority of feedback received centered around the test specification language. Recommendations suggested were evaluated and basis of estimates generated. Two major changes were incorporated. These were to allow Test Macros to reference test parameters of an entry in the Electrical Performance Characteristics Table (or specify the parameters within the macro), and to reformat Table III.



TISSS INPUT FORMAT (TIF) IS THE FORMAT FOR DESIGN AND TEST DATA INTERCHANGE AMONG TISSS SITES. A SUBSET OF TIF, THE TISSS OUTPUT FORMAT (TOF), IS USED TO PROTECT PROPRIETARY INFORMATION.

FIGURE 2.47 TISSS OPERATIONAL SCENARIO

The second Industry Review was held in Melbourne, Florida June 22-23 1988, with approximately 100 industry and Government representatives in attendance. The intent of this review was to present the Modeling Subsystem to Industry along with a process flow through the TISSS. Special emphasis was given to the Test Vector Language, TVL. A thorough presentation on TVL was given with Industry and government giving useful feedback on improvements. For the first time, CAD tools generated on the program were introduced to Industry with an opportunity to sign up for distribution.

An on-line demonstration was given at the review which included loading data into TISSS, manipulation of that data, test data generation, fault simulation, and postprocessing. The intent was to show users one scenario of TISSS usage with emphasis on test generation and simulation. The four-color TISSS brochure was provided to the attendees.

2.48 TISSS SOW Task (4.3.7) - Phase II FCA And PCA At RADC

The Functional Configuration Audit (FCA) and the Physical Configuration Audit (PCA) was conducted both in Melbourne, Florida and in Rome, New York respectively. The FCA was performed in Melbourne between 23 March 1988 and 27 April 1988 with the goal of verifying that the actual TISSS operation complied with the Software Requirements and Interface Requirements Specifications. Once FCA had been successfully completed, the TISSS was installed on the RADC computer system to demonstrate transportability. After Installation, the PCA was conducted between 14-17 June 1988 to establish the Product Baseline by examining the as-built version of the TISSS against its design documentation. In addition, the acceptance test requirements performed by the contractor Quality Assurance organization as reflected by the documents were evaluated to their adequacy for acceptance of the software.

2.49 TISSS SOW Task (4.3.8) - Phase II Final Presentation

The Phase II Final Presentation was held in Alexandria, Virginia on 12 October 1988 to summarize the TISSS software development effort and to close out action items from Test Readiness Review (TRR), Functional Configuration Audit (FCA), and Physical Configuration Audit (PCA). A preliminary draft of the TISSS Final Report was given to the RADC reviewers to critique. This critique proved quite helpful as the RADC reviewers were able to question areas that required clarification to a reader unfamiliar with the TISSS.

2.50 TISSS SOW Task (4.3.9) - ECLIPSE Kick-off Meeting

An ECLIPSE kick-off meeting was scheduled for August 13, 1987 at RADC, but was delayed by the Government until September 17, 1987. The technical issues were reviewed and discussed. The meeting was attended by RADC personnel only.

2.51 TISSS SOW Task (4.3.10) - ECLIPSE Technical Interchange Meeting

The ECLIPSE Technical Interchange Meeting was held on February 22 through 24, 1988, at Melbourne, Florida. Approximately 120 Government and industry personnel were present. Topics discussed included: the MASA program, ATF Avionics and JIAWG information, VHSIC TISSS and ECLIPSE evolution, ECLIPSE scope, objectives, status, and methodology, ECLIPSE operational concept, ECLIPSE engineering approach and functional model, ECLIPSE requirements, design capture and validation, TPS generation, LRM certification, logistics support, ECLIPSE information model, communications and security considerations, data standards, and ECLIPSE program plans and summary.

2.52 TISSS SOW Task (4.3.11) - LRM Standards Meeting Participation

This task was redefined as attending the MASA Open Forum, December 1-3, 1987, in Dayton, Ohio. One Harris ECLIPSE technical representative was present. Harris prepared presentation materials for the Government to use in briefing the forum on the ECLIPSE activity, and supported the working group discussions and side meetings during the meeting.

2.53 TISSS SOW Task (4.3.12) - ECLIPSE System Requirements Review

The ECLIPSE System Requirements Review was held on May 3-5, 1988, at Dayton, Ohio. Approximately 100 Government and industry personnel were present. Topics discussed included: VHSIC TISSS and ECLIPSE evolution, ECLIPSE goals, objectives, status, and methodology, ECLIPSE research and mission analysis, ECLIPSE operational concept, ECLIPSE engineering methodology, ECLIPSE functional model, ECLIPSE requirements and interfaces, ECLIPSE information model, CALS Initiative (presented by Air Force personnel), ECLIPSE host evaluation, ECLIPSE tools evaluation, VHSIC TISSS evaluation, data standards, and other ECLIPSE system requirements.

2.54 TISSS SOW Task (4.3.13) - ECLIPSE Final Government Meeting

A final Government meeting was held on May 5, 1988, at Dayton, Ohio, immediately following the ECLIPSE System Requirements Review.

2.55 TISSS SOW Task (4.4) - VHDL/TVL Validation Software (Option)

This option was not selected so no work was performed on this task.

3 TISSS MASTER SCHEDULE

Figure 3.0 shows the master schedule for the activities that were performed on the TISSS Phase II program.

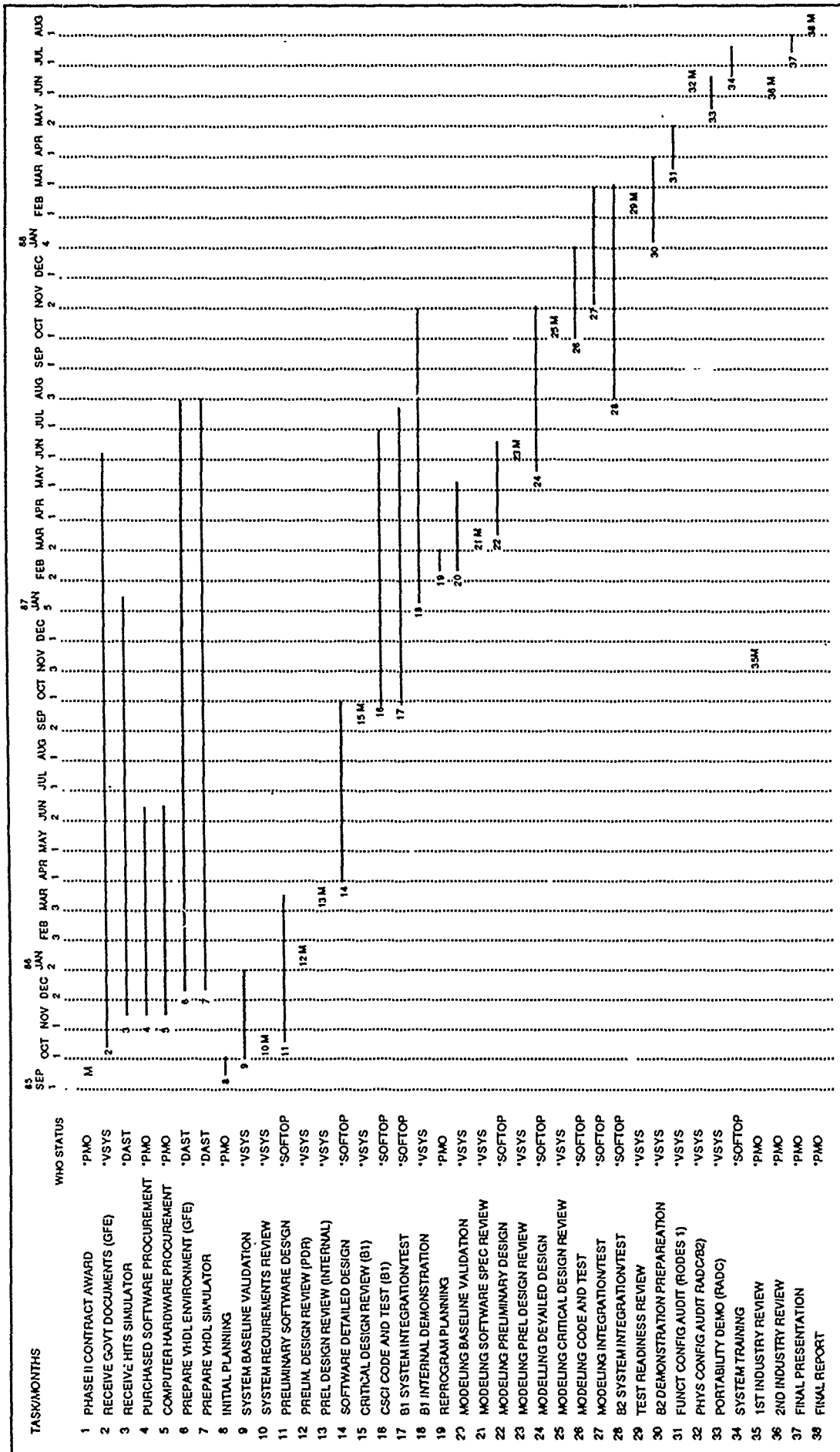


FIGURE 3.0 TISS PHASE II MASTER SCHEDULE

4 TECHNOLOGICAL INNOVATIONS

The TISSS program required a team effort between the multiple disciplines of Software Engineering, Test Engineering and Simulation Engineering. The success of the program is due to this teamwork and the technological innovations that grew out of teamwork.

4.1 Large Ada Program With DOD-STD-SDS Methodology

During the time period of the TISSS project there had been a debate about the way the Ada language and DOD-STD-SDS (later, MIL-STD-2167) could be used together to produce a synergistic result. TISSS was one of the first projects of significant size to use Ada as the implementation language and DOD-STD-SDS as the governing military standard. Most of the lead software personnel on the TISSS project brought a background of software development under government standards and a majority of the engineers had some experience with the Ada Language but no one had experience with Ada and DOD-STD-SDS together.

4.1.1 Standards Outlining The Ada And DOD-STD-SDS Process

The Software Standards and Procedures Manual created for the TISSS project defined the requirements and products of each phase of the development as they related to Ada and DOD-STD-SDS. The definitions of a Top Level Computer Software Component (TLCSC), a Lower Level CSC (LLCSC) and a Unit were also identified. These definitions are important because DOD-STD-SDS specified TLCSCs, CSCs and units in terms of functional requirements whereas Ada is designed to utilize Object Oriented Design (OOD) techniques.

The SSPM also outlined the development directory structure, methodology, PDL standards, coding standards, naming conventions, Software Development File requirements, comment headers, diagram conventions and DOD-STD-SDS templates. These items were fully defined so that personnel transitions during the project would have a minimal training impact and that consistent, usable documentation was a product of the design effort.

4.1.2 Applying The Ada Structure To DOD-STD-SDS

The development team had experience using government functional development specifications while also possessing skills in small and moderate sized Ada projects utilizing Ada methodologies without the guidance of a DOD-STD-SDS requirement. Therefore, the first concern was to integrate Ada methodologies with the DOD-STD-SDS Specification. The selected approach was to develop Ada software using Object Oriented Design (OOD) and functional methodologies. Next, a mapping of the design into the functional definitions of TLCSC, LLCSC and units was completed. The Ada design was used to create functional Yourdon charts depicting the hierarchical or operational calling structure of the software design. This approach to integrating Ada with DOD-STD-SDS produced a functional or hierarchical document from an Ada design

and methodology.

The next hurdle was with Ada packages within a CSCI. The solution was to place the packages in a logical order in an appendix of the design documents, but this approach hindered the readers ability to review a CSC or unit without having to refer to the appendices for pertinent information.

These difficulties with Ada and DOD-STD-SDS prompted the formation of a group to resolve these issues for TISSS, TISSS follow-on and other Ada/DOD-STD-SDS projects. This group would review lessons learned memos and suggestions and would continually evolve the SSPM to better deal with Ada documentation issues.

The latest approaches were used on Build 2 for the TISSS project. These methods state that a TLCSC which satisfies several logically cohesive functional, interface or performance requirements logically cohesive would include the description of the Ada package which encompasses the requirements of the TLCSC. The LLCSC which satisfies a subset of the TLCSC requirements would include the description of the Ada package that implements the LLCSC. The units that satisfy individual requirements would still be physical entities implemented in code, namely subprograms, tasks and instantiations of generic subprograms. Units may contain embedded subprograms, tasks or generic instantiations that support the unit's required operation. These embedded modules would be undocumented as separate units but would be included in the section of the unit being supported.

This approach to software development provided better project tracking by correlating DOD-STD-SDS definitions with physical products. The engineer could assess the progress and traceability of the requirements he was responsible for implementing. These definitions insure that the design was fully developed prior to the coding phase.

4.2 Hybrid Data Base

Many applications require a combination of capabilities offered by a DBMS and host file system. There is a whole class of newer applications which deal with engineering designs, where massive values of unstructured or loosely structured information must be organized and managed. The TISSS System implemented by Harris Corporation is an example of a hybrid database for an engineering design and test application. TISSS is written in Ada but utilizes the services of the ORACLE RDBMS and the VAX/VMS file system.

4.2.1 The Requirements For A Hybrid Database

Many applications demand services which extend beyond those offered by conventional Database Management Systems. In particular, applications dealing with engineering designs demand both the services offered by Database Management Systems and those offered by file systems, plus some extensions not offered by either.

Some of the requirements of design applications which are not met by conventional DBMSs are:

- a. Efficient support for extremely long objects (10KB - 100MB and even longer).
- b. Effective support for extremely complex and dynamic structural relationships (massive and dynamic schemata).
- c. Archive Capabilities - Designs will often exceed available on-line storage. A convenient way to archive and restore a specified design is needed.
- d. Version history - the ability to roll a complex object back to a previous state.
- e. Support for long transactions, which may extend over days or weeks.
- f. Support for specialized Change Control - Engineering Designs commonly undergo a series of certification steps, where each level of certification implies a different set of rules about what operations are legal against a complex object.

On the other hand, these applications also require all the conveniences and services of a DBMS, such as support of multiple concurrent users, flexible query and reporting, transaction control, database recovery, security, etc. In addition, the productivity gains which Relational DBMSs have made possible by isolating applications from underlying data structures and storage schemes should be made available to engineering design applications.

Currently, a great deal of research is being devoted to the solution of these problems. However, it will be many years before these research products reach the level of maturity, power, and robustness necessary to support production engineering information systems.

In the meantime, one approach which has been used successfully at Harris Corporation is that of the hybrid database. In such an implementation, a monitor or administration subsystem is positioned between the applications and the hybrid database. The monitor binds the DBMS and the file systems together, and

presents the user or client application with the appearance of a single integrated database. The implementation details of what information resides in each component is hidden from the user to the maximum possible extent. Instead of a set of file I/O and SQL commands, the client application is presented with a more abstract set of commands which allow him to manipulate "information objects" in a manner which is meaningful to the application, without regard to the underlying storage scheme.

Another consideration is that the functional requirements analysis and preliminary design steps are even more critical than usual. The analyst must determine the appropriate level of abstraction, not only for the static data model (structural semantics), but also for the operations on the data objects (behavioral semantics). It is not practical or desirable, as in a classical Information Engineering exercise, to reduce the data entities to third or fourth normal form. An attempt to normalize the design and test information for even a simple Integrated Circuit would yield thousands of relations, with tens of thousands of attributes. In addition, a schema based on such a model would be extremely unstable and dynamic. The analyst must therefore decide where the data model stops "caring" about the structure of the data. Similarly, it would be impractical to provide every application with a customized set of operations. The analyst must determine that subset of operations which meets the needs of the greatest number of users. He must then establish all the underlying behavior associated with each operation so that users do not conflict with each other, and the integrity of the database is maintained. For each operation, the locking and security semantics must be determined; i.e., the rules of object behavior must be determined and enforced for each different level of certification and user authorization.

4.2.2 Design Issues For The TISSS Administration Subsystem

The Administration subsystem encapsulates the ORACLE relational database and VMS file portions of the database. The view presented to client applications is that of objects and operations on those objects. The major objects of interest to applications of the TISSS are: Devices (ICs), Versions of Devices, and Selectable/Replaceable Units (SRUs) which contain descriptive information pertaining to the design and test of a particular Version. SRUs may be accessed independently or organized into Groups for application convenience. Applications manipulate the objects (Devices, Versions, SRUs and Groups) through a set of commands, such as Copy, Set Certification Status, Retrieve for Update, Roll Back to Previous Revision, etc. The action taken on each command is dependent on:

- a. The type of Operand (the object being operated on)

- b. The User's Authority to perform the Operation against this Object
- c. The Certification Level of the Object (either directly or by inheritance)
- d. Whether the object is currently in use (either directly or by inheritance)
- e. Whether the object is on-line or archived

Providing this degree of transparency to storage media has proven to be a non-trivial technical problem. In TISSS, the Administration Subsystem (Admin) is only concerned with controlling VMS files as typed objects; except for general knowledge of classes and states it is unconcerned with the content of the files. This allows the Admin Subsystem to treat files as objects to be managed, regardless of size or internal representation. Using this approach allows generic code to manage all file types. This same approach is applied in layered networking applications, where the lower levels deal with the actual network control details and the data content is uninterpreted.

The Admin Subsystem provides access to a set of objects through operations defined on those objects. In doing this, an Ada-like view of the database is presented to the application program. For example, when a request to create a SRU is made, Admin will translate the request into operations on ORACLE data and operations on VMS files. The entire process is managed as a logical transaction. Transactions are defined as operations on objects. Thus if the operations Copy, Create, Delete, and Replace are defined on an SRU, four transactions would be created; one to handle each operation. In this way, Admin appears to be a set of Ada packages, adhering to the Object-Oriented Design (OOD) methodology.

4.2.3 Data Integrity

During transaction processing, Admin must maintain consistency between the two components of the database - ORACLE control data (called the ORACLE portion) and the VMS files (called the VMS portion). To provide this consistency, the Admin system was designed so that the database would be consistent after each transaction. This relieves application programs from concern about data consistency, and therefore simplifies the design and development of applications.

Since the database must be consistent after each transaction, the processing within each transaction is very important. Admin must maintain certain relationships between ORACLE control records and VMS files while preserving the relationships between ORACLE

control records themselves. For example: An SRU must both "own" a file and must be associated with a Version (of a Device).

4.2.4 Data Protection

This section deals with protecting the hybrid database from unauthorized access. ORACLE provides excellent security features through 'GRANT's. Column level access can be restricted in this way, providing complete vertical security. But how can horizontal access be limited, since access to SRUs must be granted on a per user basis? Harris concluded that ORACLE security was not enough for the complex requirements of security in a Hybrid Database.

Further analysis led to the conclusion that VMS security was also inadequate to fill in the gaps in ORACLE security. VMS security is used to prevent casual users from perusing the VMS portion of the database. Using VMS security to prevent access from within Admin would defeat the data model completely, since Admin does not permit ANY control information to be kept with a VMS file.

The solution was to create a hybrid security system. Privileges or levels are associated with objects in the database as well as with each user in the system. Admin must validate each access to data during the beginning of each transaction based on the values of these privileges (or levels).

However, this security scheme was still insufficient, since the Admin system had to provide a SQL interface to the control records for any user to use. This access is provided by using ORACLE security. A series of ORACLE accounts is created, each having certain 'GRANT's on views of the real control information. When a DO SQL transaction is processed, Admin uses the appropriate "Unprivileged" account to provide the read-only SQL access. By manipulating ORACLE Accounts in this manner, the user has the full power of the select SQL statement, the database is protected from unauthorized changes, and Admin is relieved of the necessity to scan the incoming SQL for potentially dangerous SQL statements.

4.2.5 Transaction Logging

One requirement of TISSS is to log all transactions against the database, including the success (or failure) message for that transaction. First, ORACLE Auditing was explored. ORACLE provides extensive auditing based on user name (or user ID). While this Audit would show tables accessed, it would disallow Admin to insert information regarding VMS operations. An alternative approach was devised in which several logging tables were defined. One table holds all requests, the other holds the results, both intermediate and final. This solution allow Admin

to insert VMS errors and ORACLE errors side by side in log tables, and also allows easy identification of incompleting transactions. In the unlikely event that the log record cannot be inserted into ORACLE, Admin logs the error to a special dump file.

4.2.6 Locking

In addition to ORACLE locking, the TISSS Database requires several types of locking mechanisms. Locks that are used during a transaction to prevent concurrency conflicts are called DYNAMIC LOCKS. Locks used to check-out SRUs for long transactions are called STATIC LOCKS. Change control locks for setting and enforcing Certification states are called PERMANENT LOCKS. All locks in the TISSS database are enforced hierarchically - that is, locks on a higher-level object are inherited by the lower-level objects.

4.3 Product And Test Specification Language

Three major interfaces were required for the TISSS program in order to capture product and test specifications. Two interfaces were based on the VHSIC Hardware Description Language (VHDL), while the third interface required a completely new specification.

The program requirement was to develop interface specifications for three main data entities that formed a part of a product specification in TISSS. These are model descriptions, both behavioral and structural, for reprourement as well as validation and certification of a device; test vectors containing stimulus and response; and test specifications for MIL-M-38510 qualification and testing of a device.

Model descriptions to be entered into TISSS must follow syntax and semantic rules of VHDL. A complete specification had to be developed for both behavioral and structural descriptions. As each VHDL specification evolved, it was necessary to prove the specification by analyzing, model generating, and finally simulating some known device. The device was completely captured in VHDL following the rules imposed by the TISSS specification. Simulation output was compared against output from other simulation environments to ensure design intent and proper representation of functionality.

The major area of concern was in the specification for structural models. Structural models are required for fault simulation in order to validate the fault coverage requirement for a device. A VHDL fault simulator was unavailable. Therefore, it was necessary to take a VHDL structural description entered into TISSS and translate it to either HITS or HILO-3 modeling language. Due to the expressive nature of VHDL, a simple mapping was impossible. Functions expressible in VHDL could not easily

be expressed in either HITS or HILO-3. VHDL for structural models therefore, had to be restricted. This resulted in defining a basic set of primitives that could easily be translated between HITS and HILO-3. For each primitive it was necessary to develop the interface declaration, architectural body description, and associated timing characteristics.

At the device level it was necessary to specify syntax that would be recognizable to translation software. The syntax had to be such that a device could be simulatable once it became a component of a VHDL test bench. Complete specifications for behavioral and structural models are located in the Interface Requirement Specification (IRS), Software User's Manual (SUM), and Software Requirement Specification (SRS) for Modeling Interface.

One significant contribution by Harris in the area of VHDL was to create a Test Vector Language (TVL). The top level requirements for the language specification was: the language should be a subset of VHDL 7.2; it should be concise enough to serve as a standard; and it should be easily translatable to a variety of simulators and AMTEs.

It was necessary for TVL to fulfill requirements of both simulators and testers. TVL describes stimulus and response, time of transitions, strobe windows, and accuracies for timing measurements. It is simulator and tester independent and therefore is not biased towards any test equipment or simulation environment. It is a subset of VHDL 7.2. The current version of TVL in TISSS took on a flattened tabular format. Vectors are represented sequentially in time order. Emphasis lately has been to upgrade TVL to a more hierarchical structure and to IEEE 1076. The data content of TVL is for the most part inclusive but its representation could be enhanced. Effort is proceeding in that direction with a working group established to involve government and industry to develop a TVL that will eventually become a proposed IEEE standard.

The plans to upgrade the TVL may be found in a memo titled "System Engineering Plan for TVL Enhancement" dated 18 July, 1988, revised 16 September 1988, which is available from RADC.

TVL and model descriptions are based on VHDL. There is one other interface to TISSS that was developed using an Ada like syntax rather than VHDL. A language had to be chosen or developed that could provide TISSS with an easily representable and understandable test specification.

VHDL and ATLAS were evaluated as two possibilities. Other languages evaluated were too specific towards particular test equipment. ATLAS was too comprehensive and contained difficult parsing constructs. VHDL would have been a good choice but not much was known about the language at the time. The one concern was the rigid record structure imposed. VHDL supports no variant records, a required feature.

It was decided that a new language similar in Ada constructs would be developed. The entire software team would have Ada experience thus diminishing the learning curve. Development of parsers and tools could be expedited.

The Test Description Language (TDL) was developed along with the TDL toolbox. The toolbox parses TDL into internal data structures. The TDL toolbox is common tool used by many TISSS subsystems.

TDL is a very focused language that was developed for two top level purposes. First, TDL was developed as a means of specifying test requirements for a particular class and technology of devices, and secondly, to capture test descriptions for a particular device. In TISSS, the specification of test requirements is the Test Philosophy. A test philosophy is developed for a classification of devices. Multiple data files make up a test philosophy. However, only one test philosophy is specified in TDL. This is the Test Plan Requirement (TPR). The TPR defines the philosophy for testing a particular type of device. It describes what tests should be performed in order to qualify that device.

A test description is developed for one particular device. Multiple data files make up a test description but only the Test Plan Build (TPB) is specified in TDL. The TPB describes how the testing will be carried out for a particular device. The TPB must correspond to the TPR. It is the instantiation of the TPR for one particular device.

In TISSS, VHDL and TDL play an active role in interface specifications. These languages are necessary in order to fully capture product and test specifications for a device entered into the TISSS. Representations such as these are required in order that data can be validated, certified and finally qualified. This process ensures completeness and accuracy of data required for today's complex integrated circuits.

4.4 Automated Postprocessing

The key to Postprocessor operation is the input template code. Template code consists of plaintext interspersed with directives. Directives are identified by a prefix character and obey a defined syntax. Plaintext is any sequence of zero or more characters that does not contain the directive prefix character.

The Postprocessor expands template code by copying plaintext verbatim to the output until a directive is reached. Directives are not copied to the output but are treated as instructions to the Postprocessor. However, a directive may generate text that is written to the output. A directive may also result in a change in control of the Postprocessor, for example, causing the Postprocessor to read from a different piece of template code, or write to a different output. Once the directive has been executed, the Postprocessor resumes copying plaintext to the

output, waiting for another directive. This process is shown in Figure 4.4 in flowchart form.

The Postprocessor is able to generate a test program because some directives operate on data from the test specification. Some directives allow this information to be inserted into the output, while others guide Postprocessor expansion. For example, a directive might cause a voltage from the test specification to be written to the output. The surrounding plaintext, when processed as a test program, might instruct a tester to set a power supply to the test specification voltage.

The most important characteristic of the Postprocessor is that there are no restrictions placed on the plaintext. Furthermore, directives that develop output, generate the smallest possible amount of output, so as to reduce the bias towards a particular format or syntax for the output. As a result, the Postprocessor can generate GR-18 test programs, tabular reports, and VAX VMS DCL command procedures with equal ease.

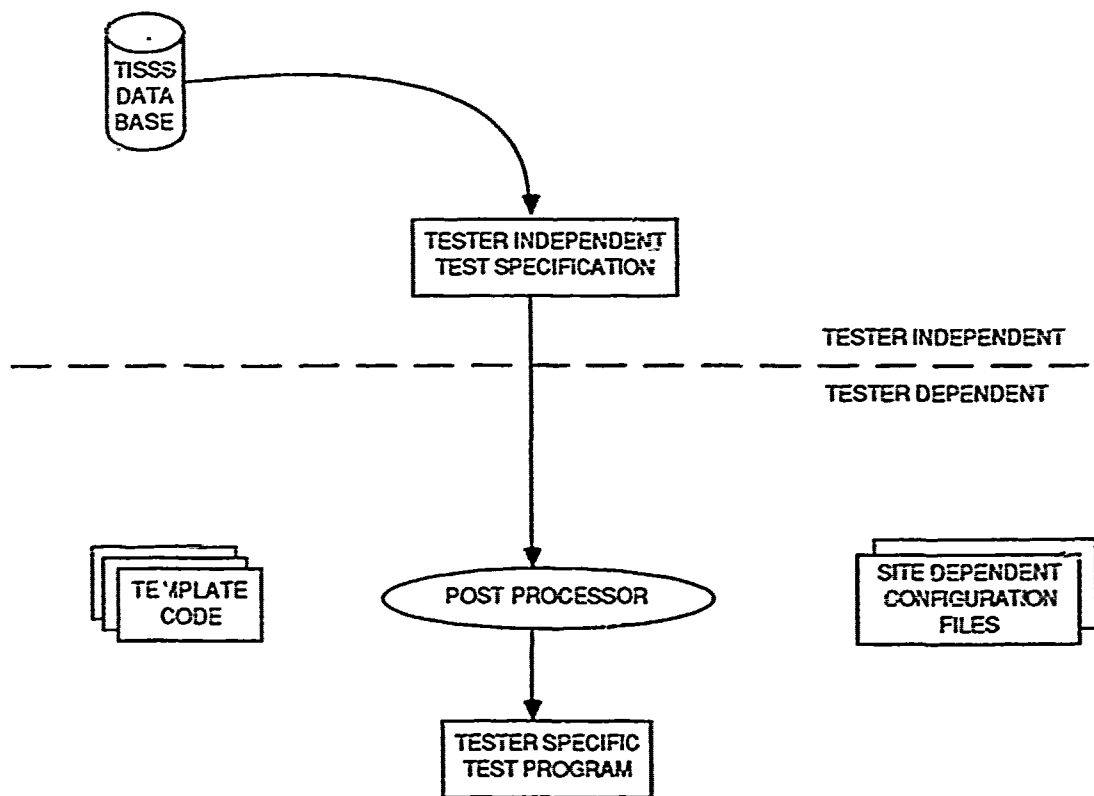


FIGURE 4.4 POSTPROCESSOR DATA FLOW

5 TISSS LESSONS LEARNED

TISSS was one of the first large Ada programs undertaken by the contractor and the first program to follow the MIL-STD-2167 development methodology with Ada. Many lessons were learned from the software development effort. These lessons are founded on the knowledge base of the personnel involved in performing the tasks and may differ from lessons learned at other companies, based on the different work environment and the software development methods used. Some of the ideas expressed here may offer nothing new to some people, but bear repeating, as these observations may be useful to those undertaking a similar project. The Lessons Learned described here are alphabetically organized into twelve categories - Common Code, Communications, Configuration Management, DID Tailoring, Documentation, Line-of-Code (LOC), Methods, Metrics, Personnel, Prototyping, Schedules, and Tools.

5.1 COMMON CODE

1. COTS and Host Interfaces. An effort was made in the preliminary design phase to of TISSS to reuse Commercial Off-the-Shelf (COTS) software and host operating system utilities and tools wherever possible to accomplish functionality. This Design-to-Cost approach was beneficial in that development costs were reduced and it enabled the system to be completed on schedule. For a single system development, this is the recommended approach. However, with the broad community of TISSS users, this method may have some drawbacks. COTS usage ties the system to a particular computer platform which may be unavailable to all potential users, and it requires that the users purchase licences and rights to products they may not otherwise want. Platform portability is an issue being studied for Line Replacable Module (LRM) extensions to TISSS.
2. Reusable software. In order to achieve cost reductions through the use of reuseable software, an early agreement should be reached between the contractor and the government on the procedure of how to handle the documentation and the system/interface testing of the reused software.
3. Common code. Creation of "common code" proved to be a method for improving the productivity of the software development team. Common code is code that performs a function used by several areas within the system. For example, in TISSS, common code was used across CSCIs or subsystems and across TLCSC's within a CSCI. Observations concerning the use of common code are:
 - a. Potential areas for the application of common code seemed to surface when an object-oriented design methodology was followed.

- b. Since the chief programmer is in a position to recognize commonality across CSCI boundaries by viewing the system as a whole, he/she should encourage common code development among all software development team members to achieve maximum benefit.
- c. Within the TISSS development team there was often disincentive to communicate or to share common code between different cost accounts. The cost account leaders were responsible for managing cost and schedule so were disinclined to assume additional responsibility to code and test common code, especially if small changes were needed to make the item more general than specific for a particular CSCI. Common code responsibility should be assigned when identified and budgets and schedules adjusted accordingly.
- d. Maximum code reuseability was not achieved in TISSS because so much common code is scattered throughout the system. In one particular case, for example, the interfaces to VMS were implemented in several CSCIs, each time in a slightly different manner. This situation could have been avoided if a separate CSCI had been defined for common code and utilities, so that a common interface could have been enforced.
- e. The CSCI team that developed the common code was responsible for its documentation, while the CSCIs that used the common code simply referenced the documentation. The overall system documentation understandability could have been enhanced by having all common code and utilities in a separate CSCI, thereby collecting all common code documentation in one place.

5.2 COMMUNICATIONS

1. Communication on a large project such as TISSS is critical to both cost and schedule. Electronic mail was the most effective tool for quick dissemination of information to many people. Without this tool, project success would have been placed in jeopardy.
2. Communication between project management and the system developers was critical to the success of the program. Not only were internal reviews scheduled prior to major customer meetings, but also at the start of each phase of the software development. The objective was to review the plan in detail and to make minor course corrections based on customer direction and cost and schedule considerations. Weekly meetings were held throughout the term of the project to review progress against the plan and to correct any deviations through a corrective action on every plan. This

bi-lateral communication channel provided early feedback to management and gave the developers an appreciation of the system as a whole.

3. Projectizing is defined as organizing all software project personnel under one functional manager. This management technique helped during the TISSS project by reducing intra-group conflict between various managers who desired to promote their own interests over those of the project. Co-locating all team members improved not only the communications but also, fostered a feeling of "esprit de corps" toward the TISSS project.
4. Each phase of the development cycle should be started with a kickoff meeting where such topics as the objectives of the next phase, the methodology, the schedules and responsibilities, etc., are stated. During Build 1, the chief programmer met only with the Cost Account Leaders to discuss these topics. This sometimes resulted in unclear direction to the developers so in the Build 2 phase, the entire TISSS team was invited to the kickoff meetings to ensure uniformity. Meeting attendees should be made aware of the importance of the information and minutes should be quickly distributed to confirm conclusions.
5. A System Engineering mini-course was held by the Software System Engineering Manager at the start of the requirements definition phase of Build 2. This mini-course proved quite helpful to engineers with diverse backgrounds by providing the expectations and techniques necessary to perform system engineering tasks.
6. The contractor's internal review teams for formal reviews such as PDR and CDR proved largely ineffective with people from outside the project. Mainly, the amount of time necessary to familiarize themselves with the project was insufficiently allocated. For complex projects such as TISSS, reviewers should be limited in number and chosen from people performing similar work.

5.3 CONFIGURATION MANAGEMENT

1. Configuration Management (CM) should be established on a large project early in the requirements definition phase in order to develop CM procedures, to establish the CM account and library structure, and to exercise control over the documentation through Change Control Board (CCB) meetings. After every major phase of the project, the developmental documents should be baselined into the CM library. This procedure proved to be very beneficial on TISSS as documentation revisions were performed in a controlled way, thus preserving the changes and allowing traceability to previous versions.

2. Configuration Management was able to operate with inexpensive, non-integrated DEC tools such as the Ada Compilation System (ACS) and the Code Management System (CMS). Both tools performed very well and have actually been improved since the start of the project. The CM computer account contained over 70 different CMS libraries and 17 Ada libraries, organized into a three-level hierarchy. In addition, over 90 DCL command procedures were written to automate the steps involved using these tools. The command files provided a log of CM actions which were attached to the Internal Software Delivery Form (ISDF) and reviewed by the CCB. These command files were important for CM productivity, throughput and repeatability. More integrated and capable CM tools are available today that could make the CM job even more productive.
3. The Configuration Management task was budgeted to support one CM clerk full time for the TISSS development. This level alone was insufficient for such a large Ada project. Engineering expertise was required to establish the CM library structure, to establish the ACS library structure, and to coordinate the initial compile and link command procedures. Engineering assistance was also needed to develop other command procedures to automate the CM tasks.
4. Many of the compilation changes and the rebuilding of the CM ACS libraries during developmental modifications were very lengthy and were best performed during off-hours. This approach allowed for better system resource utilization during the day and avoided the problem of engineers working with changing or obsolete libraries. The CM engineer should be able to work late hours or have access to a remote terminal at home since command procedures often require checking, correcting, and/or restarting at odd hours to assure successful completion.
5. CMS libraries should be named so their content and purpose are obvious. The naming convention chosen for TISSS is clear and easy to follow. Sometimes it was difficult for new people coming on to the project or to those who were unfamiliar with the CM library, to know where the files were located. A chart showing the account structure was developed mid-way through the project to help those people. Perhaps an updated formal release of the CM Procedures and Practices Manual at this point would have been useful. The government recognized the need for this information, so the account structure was included in the Version Description Document (VDD) delivered at the end of the project.
6. When changed documents were delivered to CM, it is important to quickly rebuild the documents into the proper output format (using Runoff) so that they could be put into a current copy directory accessible by everyone. In this way, an up-to-date on-line copy of each document was always available. Individuals could either print their own copy or

review the document on-line. The entire team was notified of document changes by electronic mail.

7. The Program Trouble Report (PTR) and Internal Software Delivery Form (ISDF) provide the basic information required by the Software CM plan to affect changes to the system. The forms were revised at least twice during the TISSS project, each time tailoring them with more project specific and useful information. Initially, the forms were printed on paper and filled out by hand. The latest versions were stored electronically and edited on-line by the person submitting the change. By using electronic forms, the readability of the information was improved and led to a more completely filled in form. A still better mechanism would be to create a tool on top of a data base for storing the PTR/ISDF information. This technique could provide better facilities for tracking the status and closure of the PTRs.
8. Configuration Management difficulties seemed to be related to the lack of understanding by the TISSS developers in proper CM procedures and in fully completing the PTR and ISDF forms. A more comprehensively written CM plan could have aided in this understanding. As the procedures evolved, the CM plan should have been revised to help the developers learn the procedures. Often, the PTR and ISDF forms contained insufficient information to completely describe the problem or the solution. Sometimes only a portion of the PTR could be closed by the ISDF, leaving the balance of the change incomplete and causing difficulties with the tracking and cross-referencing of PTR's in the CM filing system. A better way would have been to close the PTR and issue a new but similar one with reduced scope. In this way the tracking process would have been cleaner. Also, one ISDF was sometimes used to deliver several files to CM and to close several PTR's. While this was certainly a fast method of closing PTR's, it became difficult to trace the effect of an individual PTR at a later date. Therefore, the number of PTR's closed by a single ISDF should be kept to a small number, preferably only one per PTR. In order to make the CM PTR/ISDF system fast and efficient, and to prepare CCB agendas and status reports, the system should be automated as much as possible with a data base centered tool.

5.4 DID TAILORING

1. The Data Item Descriptions (DIDs) that were called out in the TISSS Contract Data Requirements List (CDRL) were extracted from the draft version of DOD-STD-SDS, dated 5 December 1983. These DIDs were mostly based on using older languages such as Fortran, and failed to address concepts that were used in documenting an Ada design. The DID's were written very generally to apply to all types of software programs and some sections were not applicable to the type of software system

being developed. For example, TISSS is a data base system that is supported by tools and a user interface, not an embedded system with several hardware interfaces and stringent timing requirements. Thus, those sections in a DID that dealt with interrupt handling and timing requirements were unnecessary for TISSS.

2. DID tailoring is a mechanism to provide only the necessary documentation for a system and to present it in the most meaningful way. DID tailoring should be negotiated prior to contract award, if possible. Some additional DID tailoring may be identified throughout the development process. All changes should be communicated to interested parties, including the customer, the Program Office, any Independent Validation and Verification (IV&V) organization(s), internal Software Quality Assurance (SQA), Contracts, as well as the software development team.
3. The updated MIL-STD-2167A, Defense System Software Development, dated 29 February 1988, encourages tailoring of the Data Item Descriptions (DID's) on a per contract basis. The standard defines tailoring as the elimination of non-applicable paragraphs or subparagraphs denoting the deletion with a statement following the deleted paragraph heading. For TISSS, DID tailoring was performed by the software engineers in the preparation of the Software Top Level Design Document (STLDD) and the Software Detailed Design Document (SDDD) as well as other documents. The customer was informed of DID variances as they were discovered and he provided verbal direction with subsequent written follow-up of the agreement. However, these agreements were not always timely and SQA and IV&V contractors were not always aware of their existence, and many hours were unduly spent trying to explain and justify DID variations to both groups. Early DID tailoring approval prior to contract award would have avoided this time consuming and inefficient process.
4. From the contractor's experience in working with MIL-STD-2167 on this project, it was beneficial to both the customer and the contractor to include additional information in subsections or appendices not called out by the 2167 standard. Also, it was advantageous to modify the content, detail and format of the information required by certain areas in the DIDs, consistent with the general direction of the military standard.
5. Document contents and the physical preparation of the documents should be automated as much as possible. Document style/format should be exactly what is produced by a tool, within adjustments provided by that tool, and by mutual agreement between the customer and the contractor. The productivity gained by using the tool in the first place is diminished if the document requires postprocessing or editing after tool output.

6. The number of documents required by the military standard will be reduced by keeping the number of CSCIs to a minimum. Contractor developed software should be separated in the static hierarchy from Commercial Off-the-Shelf software (COTS) and Government Furnished Software (GFS). The design and operation of COTS and GFS was undocumented by the contractor since COTS and GFS were documented elsewhere. However, the interfaces to COTS and GFS should be identified and described in the Interface Design Document (IDD).
7. In order to fully utilize reuseable document fragments, they should be contained in a single CMS library or have a single copy of common parts. Thus, the same paragraph text may efficiently be reused in several documents. For example, a project Glossary may be developed and for easy reference, appear in the Notes section of every document produced. When additions or modifications are made to the Glossary, the changes are made in one place and the automated regeneration of documents quickly propagates the change throughout the system.
8. A review of the style and expected content of each document should be scheduled between the customer and the contractor at the beginning of each stage for documents produced in that stage. This approach would shorten review cycles and minimize IV&V comments which criticize for style only.
9. In reference to the SDDD, Section 3.1.3, Memory and Processing Allocation, if no time critical real-time requirements or size restrictions exist, this section should be tailored out since the information is very difficult and costly to produce.
10. Again in the SDDD, automated tools should be used to derive as much information as possible from the Ada PDL or code. The information given on each input, output and local data element will be its Ada type information and descriptive comment. Information such as units, range of values, precision/resolution, legality checks, data type and data representation will not be explicitly listed since it can be derived from the Ada type information.
11. For maintenance of large software systems developed by others, it has been the contractor's experience that to obtain an understanding of the software architecture and operation, a graphical high level structure diagram was the most useful aid. When specific problems arose, the code was utilized directly, especially if it was well commented. The PDL is rarely looked at and difficult to maintain. PDL, however, is useful during design development when not all the details are known and is included in the SDDD at CDR. But as the system is implemented, the Ada PDL should be replaced in the SDDD with Ada code.

12. Design decisions and reasons for selection of alternatives and/or approaches are included in the SDDD, Section 6, Notes. Generally, software developers neglect to capture this information but for better understanding during software maintenance, this information is most useful and therefore should be documented in this section.
13. Software User's Manual (SUM) - An appendix could be included in the SUM that contains each menu screen or SQL Form that is contained in the User Interface to the system. This data would be very useful as a reference for the required execution procedures and user input described in Section 3 of the SUM.
14. Software Product Specification (SPS) - As the final version of the SDDD contains the PDL that is the actual as-built code, it is unnecessary to include the actual code listings in the SPS. Therefore, it is recommended that the SPS as a program requirement be deleted since the data already exists in the basic documents that make up the SPS.

5.5 DOCUMENTATION

1. Most of the difficulties with design documentation that occurred in Build 1 were addressed and resolved in Build 2. The Design Document Generator (DDG) tool developed internally, generated the runoff file for the SDDD directly from the PDL or code files along with a template file and configuration file as inputs. This enabled the designers to perform all their work within one file, the .ada file, which was used first as a PDL file and then as a code file. Functional descriptions of the unit were placed in the code headers. The DDG tool placed the descriptions, inputs, output, processing, etc. into the appropriate sections of the document so no references to listings were required. Also, the DDG calculated which units used other units and which data items were global and where they were used. During Build 1, this information for the SDDD was computed by hand which may have led to an inconsistency and distrust for the accuracy of the data. The standard program header was changed to accommodate the DDG tool and the code template for Ada files was made more sensible and automated.
2. Even though an object-oriented design approach may be used, it was found that for TISSS the Yourdon diagram (with conventions described in the SSPM appendix) served a useful purpose. These diagrams illustrated the calling relationships between units in a program and provided a high level graphical view of the overall design. These diagrams were also found to be very useful in maintenance tasks of tracking down bugs or by making enhancements and should therefore, be kept current with software changes in order to continue their usefulness.

3. In Build 2 of TISSS, it was beneficial to create templates for the various configurations of Yourdon diagrams (i.e. two boxes on a page; three boxes on a page; etc.). Thus, unlike Build 1, all Yourdon diagrams were layed out consistently and made much more readable. Templates also improved the efficiency with which the engineers were able to create the diagrams and the speed with which the clerks were able to draw them on the MacIntosh.
4. Yourdon diagrams should be automatically generated from the Ada PDL/Source code. The code contains the actual Ada calls and thus all of the information necessary for the Yourdon diagrams. Automating this task would save both engineering time to create the diagrams as well as clerk time to produce them. A tool in conjunction with the Design Document Generator (DDG) was created internally near the end of the TISSS contract, that parsed the Ada code and printed the resultant diagram on a laser printer using Latex. This tool proved the feasibility of the concept and thus should be used in future maintenance work.
5. Yourdon conventions for TISSS were developed during the Design phase of Build 1. A few special cases arose during that phase which caused the standards to be revisited and added to as necessary. The resultant conventions were a bit complex and one simplification was made in Build 2. The asterisk beside a box t. indicate that the unit was previously expanded was dropped.
6. TISSS documentation was completed with inadequate tools for merging text and graphics within a document. VAX Runoff was used for the text portions and the MacIntosh used for graphics figures. The graphics were manually coordinated and "cut and pasted" by hand. There are more efficient publication processing tools available today where graphics can be merged with text within the same operating environment.
7. The SDDD was made more understandable when it was organized by Ada packages as was done during Build 2. Since Ada packages were not addressed by MIL-STD-2167, the Ada packages were included within an appendix for Build 1, not in the main part of the SDDD document. The mistaken thinking at that time was that the Ada packages were not really part of the design but an after-the-fact grouping of units. It was learned through experience that the Ada packages were the key to the design and thus a natural for organization and expression in the documentation.
8. Most documents developed on a project should contain an index to some degree to aid the reader. Although the documentation preparation tool used on TISSS (Runoff) provided the capability to create an index, each individual occurrence of a word that was to be referenced in the index had to be marked with the appropriate Runoff command. Tools were

created to enhance Runoff's indexing capability by creating lists of words from the document and marking all occurrences of designated words with index commands. It was found that the Software User's Manual (SUM) and the Software Standards and Procedures Manual (SSPM) needed the most comprehensive indexes since these documents would be frequently used for reference. For the Software Detailed Design Documents (SDDD), it was found useful to index every CSC and unit and also to cross-reference each unit with the package where the unit was contained.

9. The creation of the Software User's Manual (SUM) and Computer System Operator's Manual (CSOM) would have been an easier task and of more use to the developers if these documents had been prepared in parallel with the TISSS system instead of after its completion.
10. The Program Trouble Reports (PTRs) against the requirements documents should be completed soon after they occur because these documents form the baseline for the test and development teams. For TISSS, the requirements document updates lagged behind the direction received by the developers so the documents became ineffectual as a baseline. Additional system engineering resources could have helped to alleviate this difficulty.

5.6 LINES OF CODE (LOC)

1. The TISSS software development was managed by Lines of Code (LOC). At each major software development milestone review, LOC estimates were prepared and reported on, based upon the latest knowledge of the system design. This gave managers the visibility to see where LOCs were growing or to identify potential out of scope features not specified by the requirements documents. Paying attention to the LOC estimates helped to ensure functionality compliant with the requirements and provided a criteria for decision making when evaluating alternatives. Sometimes a feature would be selected that would require more manual steps on the part of the user because it greatly reduced the LOC count. This trade off analysis provided resources to enhance system features, perhaps at the expense of user friendliness.
2. The technique of managing by LOCs had some difficulties that were discovered through experience. Initially, there was confusion on what constituted a line of code. It was agreed that the Boehm standard for LOC counting would be used to prepare LOC estimates. That is, each physical line, excluding comments and blank lines, would count as an LOC. Boehm LOC had been used as a contractor division standard in previous projects and could thus be compared to TISSS for equivalency. The LOC that was presented in the proposal and BAFO was based not on the Boehm standard but on equivalent

Fortran functionality. There was some effort spent trying to resolve the equivalent mapping factor.

3. The second area of difficulty with LOC estimation was that people on TISSS were unfamiliar with estimating LOC in Ada. Some of the differences between estimating LOC for Ada vs Pascal/FORTRAN 77 were:
 - Procedure and Function declarations often had to be repeated up to three times in Ada. Declaration may be required for the Procedure itself, as well as the Package Specification and Package Body.
 - Strong typing in Ada required development of conversion utilities. TISSS required about 4K LOC just for conversion from one type to another. These LOCs were very difficult to estimate early in the project.
 - Package Specifications and Bodies should be conceptualized very early, and LOC increased accordingly.
4. Software style standards have much influence on Boehm LOC estimations (from 25% to 100%). Style standards must be well understood when comparing projects and using project productivity data for competitively bidding new work.
5. With experience gained with using Ada, the contractor has improved his ability to use Boehm LOC estimating techniques. One may still think in terms of "functions", but use a multiplication factor to reach Boehm. The LOC estimates on TISSS Build 2 were more accurate for the resulting implementation based on the lessons learned from the Build 1 experience.

5.7 METHODS

1. The MIL-STD-2167 document known as Software Standards and Procedures Manual (SSPM) was a critical document for defining the methods and standards that the software developers were to follow on TISSS. Initially, an SSPM was delivered with the Phase II proposal. This was a very high level document that was extremely insufficient for describing the procedures necessary to perform the day-to-day software development work on TISSS. The contract required no subsequent updates or deliveries of the SSPM, but because it was such an important document, the SSPM was expanded and enhanced during the software development period for internal use.
2. The chief programmer and cost account leaders recognized early on the need for adding to the SSPM and made modifications. But since the group had no previous history with Ada and MIL-STD-2167 projects, many of the standards and

procedures were modified throughout the project as experience was gained. The resulting SSPM is now the most comprehensive standard used by any group at the contractor facility, yet it still could be improved. In most cases, difficulties with the SSPM were not discovered until one of the CSCIs reached a point in its design where a particular standard had to be applied. This resulted in many changes that in turn forced existing work to be updated. Productivity and morale were adversely impacted by such changes.

3. For future software development projects, it is imperative for program control that the methodology and standards be prototyped in the same manner as critical design areas. In fact, the standards are probably more critical (in terms of re-work needed) than for most design areas. This prototyping of development standards implies that time, money, and methodology expertise is available at the start of the project. In the case of TISSS, there was no project experience identified in the entire Sector for DOD-STD-SDS (draft version of MIL-STD-2167) or for Ada development using a government standard, much less a combination of the two. It is hoped that the experience gained on TISSS can be used profitably by others.
4. A key feature of the contractor software design methodology was that compilable Ada PDL was used. This not only made the transition from PDL to code very easy but it enabled the interfaces to the solution entities to be compiled early in the design process ensuring a concrete definition of the interface and independent development of items on either side of the interface.
5. At the completion of Build 1 of TISSS and before starting Build 2, a major rework was performed on the SSPM to incorporate lessons learned from Build 1. The code developed under Build 1 was unchanged making it incompatible with the new standard. Thus, there are two separate stand-alone SSPM documents. Some of the key changes for the Build 2 SSPM were:
 - More emphasis was placed on the use of "Object Oriented" design in both the preliminary and detailed design phases.
 - The definitions of TLCSC, LLCSC, Units and the static hierarchy as it applied to the MIL-STD-2167 documentation was clarified. LLCSC was changed to be an Ada package instead of depending upon whether they called other units.
 - A tool called the Design Document Generator (DDG), was developed to automatically generate the design documents from the PDL or code files. The standard program header was changed to accommodate the tool and the code template for Ada files was made more sensible and automated.

Procedures were changed to describe how software engineers should design the PDL and code to produce the best documentation with the tool.

- Package qualification was required of all data used within a unit that was not declared within that unit, even if it was within the enclosing package. This requirement was added for understandability as well as to work better with the DDG tool.
 - Much more stringent file naming conventions were derived for all files, not just source files but test plans, test data, test drivers and stubs and test results were included. These naming conventions were left to the individual developers or work package leaders during Build 1 and were thus, quite inconsistent across the project.
6. The time required to become proficient in Ada is tremendous. Each team member had, as a minimum, attended one (or more) Ada classes. This level of training was much more detailed and exhaustive than a normal in-house course, but did not qualify a person to design in Ada. Until more experience was obtained by the contractor, it was extremely risky to attempt a project of the magnitude of TISSS without having many experienced Ada experts on hand.
 7. Data flow diagrams prepared during the requirements phase of a software development program are very useful for the understanding of the system, as long as these diagrams impose no design entities or functions prior to the actual preliminary design task.
 8. A deficiency throughout the entire life cycle of the TISSS project was in the handling of formal customer and informal internal action items. Better mechanisms and procedures were needed to clearly capture action items that stated the intent of the initiator; better mechanisms were needed to track the status of action items opened and closed; and better methods were needed to complete and review the closure of action items. Much effort was spent trying to "find" old action items, track down incomplete action items and organize them for reference and presentation. A decision criteria should have been developed to evaluate the "worth" of completing action items. Simply because someone requested an action item, the effort to complete the task may not have been justified. Several action items were generated and worked on for TISSS which did nothing to contribute to any ongoing task or related effort. These were often extraneous or just interesting to someone. An investment needed to be made to clarify action item procedures and to automate those procedures and tracking with a database tool.

9. Because the Postprocessor (AP) CSCI was a new technology area in concept, an Operational Concepts Document (OCD) should have been written for it during the requirements definition phase, even before the Software Requirements Specification (SRS) was prepared. This would have clarified its intended purpose, enabled the customer and other experts to provide early feedback and to have allowed the SRS requirements document to be written with greater understanding.

5.8 METRICS

1. A major deficiency area during the TISSS development was formulating guidelines for the type of metrics and how they should be gathered and used to highlight problem areas for improving productivity of the software development process. The only metric that was gathered and measured where an accurate history exists was the line-of-code count.
2. Whatever metrics are required for performance measurement, a mechanism for entering data on-line and as it occurs should be developed instead of recreating the information after the fact from recall or from an incomplete history. Cost and schedule data are very important for measuring productivity and useful for accurately bidding future work. There was no mechanism for organizing and periodically capturing on-line cost and schedule information for TISSS.
3. Sizing and timing estimates for performance tracking provided useful information to guide design decisions. Sizing was performed during preliminary design once the data structures were developed. Tools were written to project the memory required for different variations of the input data, allowing the optimization of system performance. Timing measurements were performed during the coding phase in order to tune algorithms when performance deficiencies were uncovered.

5.9 PERSONNEL

1. One of the most important lessons learned on TISSS was the criticality of an extremely effective design and implementation team. The TISSS team consistently accomplished whatever was asked of them, often going well beyond what was expected. Similar challenging projects (tight budget and a fixed price contract) of implementing a system with a new technology and non-specific high-level requirements and reliance on inter-division expertise, demand high performers at every level of the team.
2. System engineering support for both software development and VHSIC test was insufficient for a project of this size. Additional system engineering effort during the Requirements

phase to more clearly state the requirements would have made the job of the software developer easier. A full-time software systems engineer should have been assigned to manage changes during the life cycle and to interpret the requirements to ensure that the design, the code, and the tests were compliant.

3. In order to avoid duplication of effort, job descriptions should be prepared early on to clarify responsibilities and relationships. The differences between the functions of the Chief Programmer and those of the Chief Engineer, for example, should be well defined and clearly stated to achieve high productivity and reduce areas for conflict.
4. It was extremely effective to have the design team involved during the Requirements phase. Not only were the designers able to be productive quickly during the Design phase, they had gained a better overall understanding of the system as a whole, which led to more effective design decision making later. Since the designers helped to write the TISSS' requirements and at the same time, were aware that they would be responsible for implementation, there was the additional motivation to make the requirements understandable and testable.
5. It was important to have the entire group within a subsystem working on the top level design in order to enhance overall design understanding. If the work was partitioned too early, the exchange of top level design ideas would have been stifled, thus limiting full understanding.
6. When a project is large enough, clerical support should report to the Cost Account Leader instead of being assigned from a support pool. There were several instances during the TISSS development when support personnel were temporarily reassigned to other projects by their supervision, resulting in an inefficient resource allocation that required engineers to perform data entry tasks. A cost account or CSCI that has over a 15K lines-of-code budget should have a clerk assigned full time.
7. In retrospect, the clerical support budget for TISSS was inadequate for the amount of documentation required by the project. Additional computer graphics operators, technical writers and general data entry people could have been used to support the engineering developers. Also, a policy of cross-training these people in all clerical task areas would have allowed higher productivity. Project management should be aware that the lack of trained clerical support on a project of this size can be very demoralizing to the engineering developers working to meet a tight schedule.
8. When assigning people to perform a small task, it was more productive to make a full time assignment for a shorter time than to assign a part time person for a longer period.

5.10 PROTOTYPING

1. The prototyping which was performed during the various TISSS design phases proved very beneficial. More prototyping was performed than for most development programs, yet in retrospect there were several areas where earlier prototyping could have helped. Much of the prototyping was performed in the Preliminary Design phase, which is generally earlier than normal. However, extensive prototyping in the Requirements Analysis phase would have been even better.
2. When a system uses embedded Commercial Off-The-Shelf (COTS) software, early prototyping of the system interfaces should be considered. This prototyping should have been completed by the end of the TISSS requirements definition phase. Even though some prototyping was performed for all TISSS embedded COTS products, it was generally inadequate because often these were the areas that caused most of the problems during test.
3. In order to help prevent changes in requirements during the design phase, extensive prototyping is recommended for newly developed tools and languages. The customer should be involved in reviewing the prototype results to plan how the knowledge gained can be used in the program structure.
4. All prototypes should be written in accordance with project coding standards so that all or portions of the prototypes can be assimilated into the final product.
5. The TISSS menus should have been designed and prototyped much earlier in the project cycle. The menu structure was completed well into the design phase after Preliminary Design Review. The Design Input menu prototype, for example, was completed half way through detailed design. Menus should be specified by Systems Engineering together with a software developer to indicate feasibility and presented as a part of the requirements. It would have been beneficial if the customer and final end-user could have provided more user specific information during the requirements phase. Tools are now available that allow rapid prototyping of user interfaces and menus that could be utilized to facilitate this task.

5.11 SCHEDULES

1. The execution of the TISSS Phase II program was planned for a very tight schedule based on fiscal and customer constraints. Considering that a high-risk, new technology was to be developed, program success and adherence to the schedule was attributed to establishing a competent, dedicated, and success motivated development team.

2. The utilization of detailed schedules, planned to daily resolution, was an extremely effective management concept. The schedules were manually statused and reviewed at least on a weekly basis, more often when needed. Direction was provided to the software developers through these daily schedules which indicated what was expected of them and the coordination with the overall plan. Automated tools to plan and produce these schedules in a readable, statusable output form would have helped increase cost account leader productivity.
3. The idle period between the time material was prepared for a review and the review itself was not allowed to be wasted. Development activities were planned such that work was started on tasks for the succeeding phase.
4. In preparing the detailed schedules, consideration should be given to the establishment of milestones for intermediate and internal reviews as well as for action item completion.
5. It was extremely difficult to perform two separate development tasks in parallel with the same development team. The Build 1 schedule for TISSS, for example, was overlapped with the Build 2 schedule in order to maintain the overall program schedule established at the outset of the program. This turned out to be very inefficient since it was difficult to complete the Build 1 tasks while trying to maintain the Build 2 schedule. A better approach would have been to focus on completing the Build 1 tasks before starting Build 2.
6. The schedule for Build 2 was even more compressed than the Build 1 schedule. Adding headcount is not always an effective solution to increased workload. Individual pieces can be completed quickly but not necessarily integrated. The coordination of the entire development task generally falls onto a few people such as the cost account leaders, chief programmer and chief systems engineer, causing them to be extremely overworked. This region of diminishing returns suggests an axiom. Despite how badly its needed, the schedule for high quality working software can be compressed only so much.

5.12 TOOLS

1. The tools used for a project can have a tremendous impact on productivity. The TISSS team developed tools for everything from template generators to Software Development Folder (SDF) creators. Literally hundreds of engineering hours were saved by extensive tool development. However, because of cost and schedule constraints, the TISSS tools tended to address the less complicated support tasks.

2. Consideration should be given during the proposal phase to the importance of establishing a budget for commercial tool purchases to aid in the software development process. Generally, management pressure forces productivity cost issues such as tool purchases to be cut because they are either not well understood or are difficult to justify on their own merit. Many engineers when faced with a large task, would rather write a tool (command file or program) to help them complete the task. It is far more efficient to purchase an off-the-shelf product that satisfies a need rather than developing it. Also, several times on the TISSS project a similar tool was developed by more than one person in parallel to solve a particular problem. This led to two and sometimes three variations of the same tool and a duplication of effort. The chief programmer should surface tool needs during the proposal phase and throughout the project to identify the productivity issues to justify the expenditure.
3. Improved productivity and efficiency should be a concern of all project team members. The management procedures for a chief programmer to obtain tools should not require an inordinate amount of effort to be accepted by productivity minded project management.
4. For projects similar to TISSS, a recommendation would be to establish a separate budget for the Chief Programmer to use for tool development. Similarly, the Chief Programmer should be responsible for maintaining and expanding the tool set, as well as ensuring that the tools developed have the broadest practical application. The people who need the tool are the most highly motivated to develop it, but in the case of a larger tool, non-project help is needed. During the TISSS negotiation phase, the tool budget was cut from the original proposed budget, forcing a reduction in the scope of the developed tools. An early identification of tool need may have placed a higher priority on tool acquisition, resulting in cost reductions in other areas. Another recommendation would be to identify and fund a "tools guru" to consolidate tool development in one place.
5. The Software Tracking System (SOFTRACS) and the Design Document Generator (DDG) were two large tools partially developed on program funds. Together, they have saved hundreds of engineering manhours and have helped to enable the project to be completed within cost and schedule. The requirements for and the guidance in the development of these tools was provided by the chief programmer.

6 STATISTICS ON COMPLETED TISSS PRODUCT

The TISSS project delivered many different documents and various forms of source and test code to its customer. This section shows the size and categorization of those deliverables as a reference in comparison with other software projects.

6.1 Human Effort

The TISSS project as a total took over 200,000 person hours to complete. A breakdown by labor category and phase is given in Table 6.1.

TABLE 6.1 TISSS LABOR HOURS

<u>LABOR CATEGORY</u>	<u>CATEGORY NO.</u>	<u>TOTAL</u>
		8078.13 (SEE NOTE)
MISCELLANEOUS	0000	10733.86 (SEE NOTE)
PRINC ENGINEER	010	36237.25
ASSO. PRIN ENGIN	012	21276.95
SR. ENGINEER	014	91329.65
ENGINEER	016	32014.75
SR TECH	022	144.00
TECHNICIAN	024	212.00
PROJ PLANNER	042	2518.65
PROJ CLERK	044	4317.27
DRAFTSMAN	062	9.00
TECH WRITER	072	2250.75
DOC SPECIALIST	074	4637.50
DOC CLERK	076	3172.15
INSPECTOR	082	2.00
E/M INSPECTOR	084	2.00
QUAL SPECIALIST	086	383.00

TOTAL		217348.91

NOTE:

THE HOURS INCURRED INCLUDE PEOPLE WHO WERE REQUIRED TO PERFORM TISSS WORK AND HAD INDIRECT JOB CLASSIFICATION STATUS. EXAMPLES OF THESE JOB FUNCTIONS ARE: CONSULTANTS, INDIRECT SECRETARIES AND CLERKS, AND PROCUREMENT PEOPLE.

6.2 Documentation Page Count

The following documents were printed and delivered to RADC for TISSS. The CDRL number is shown for each document and well as the number of pages in the document. The Final Report is not included in the total page count.

TISSS Configuration Management Page Count

1 September 1988

CDRL	Document	Pages
A013	S/W Standards and Procedures	86
	S/W Standards and Procedures (2)	112
A015	S/W Configuration Management Plan	21
B001	R&D Status Reports (27)	1070
B002	S/W Top Level Design Document	1189
	UI	586
	CP	32
	DI	144
	AU	83
	OU	48
	AP	80
	AD	90
	MI	126
B003	S/W Test Plan (AP)	202
B004	S/W User's Manual	273
B005	S/W Detailed Design Document	7569
	UI Volume 1	402
	UI Volume 2	440
	CP Volume 1	107
	DI Volume 1	338
	DI Volume 2	648
	DI Volume 3	333
	DI Volume 4	628
	AU Volume 1	263
	AU Volume 2	439
	AU Volume 3	373
	AU Volume 4	221
	AU Volume 5	232
	AU Volume 6	279

	OU Volume 1	448
	AP Volume 1	365
	AP Volume 2	281
	AP Volume 3	534
	AP Volume 4	347
	AD Volume 1	437
	AD Volume 3	454
	AD Volume 4	371
	AD Volume 5	304
	MI Volume 1	292
	MI Volume 2	378
	MI Volume 3	371
	MI Volume 4	355
	MI Volume 5	534
B006	Interface Design Document	106
B007	Data Base Design Document	58
B008	S/W Test Description	407
	UI Volume 1	34
	CP Volume 1	24
	DI Volume 1	75
	AU Volume 1	35
	OU Volume 1	32
	AP Volume 1	21
	AD Volume 1	94
	MI Volume 1	92
B010	S/W Test Procedure	3363
	UI Volume 1	456
	UI Volume 2	446
	CP Volume 1	76
	DI Volume 1	613
	AU Volume 1	253
	OU Volume 1	140
	AP Volume 1	360
	AD Volume 1	379
	MI Volume 1	640
B011	S/W Test Report	437
	UI Volume 1	71
	CP Volume 1	12
	DI Volume 1	66
	AU Volume 1	64
	OU Volume 1	63
	AP Volume 1	13
	AD Volume 1	76

	MI Volume 1	72
B013	Version Description Document	185
	Version Description Document	68
	Computer System's Installation Manual	117
B014	Computer System Operator's Manual	203
B015	S/W User's Manual	1374
	Volume 1	374
	Volume 2	482
	Volume 3	518
B018	Training Course Outline	29
B017	Quarterly Review	259
B021	Operational Concepts Document	242
B022	System Segment Specification	252
B023	S/W Requirements Spec	636
	VI	37
	CP	17
	DI	45
	AU	42
	OU	25
	AP	161
	AD	70
	MI	239
B024	Interface Requirements Spec	256
B029	S/W Quality Assurance Plan	84
B031	Contract Funds Report	12
B032	Work Breakdown Structure	2
B033	Cost/Schedule Status Reports	304
B034	Published VTS Manual	138
B035	Reserved VTS Manual	59
B036	Published VTS	120
B037	Reserved VTS	158

Non-CDRL Documents:

TDL Tool Box Description Specification	33
TVL Tool Box Description Specification	22
Government Operational Policy Considerations	73
Total	19,689

Figure 6.2-1 shows the relative size of the documents by category and Figure 6.2-2 shows the information by individual documents produced on TISSS.

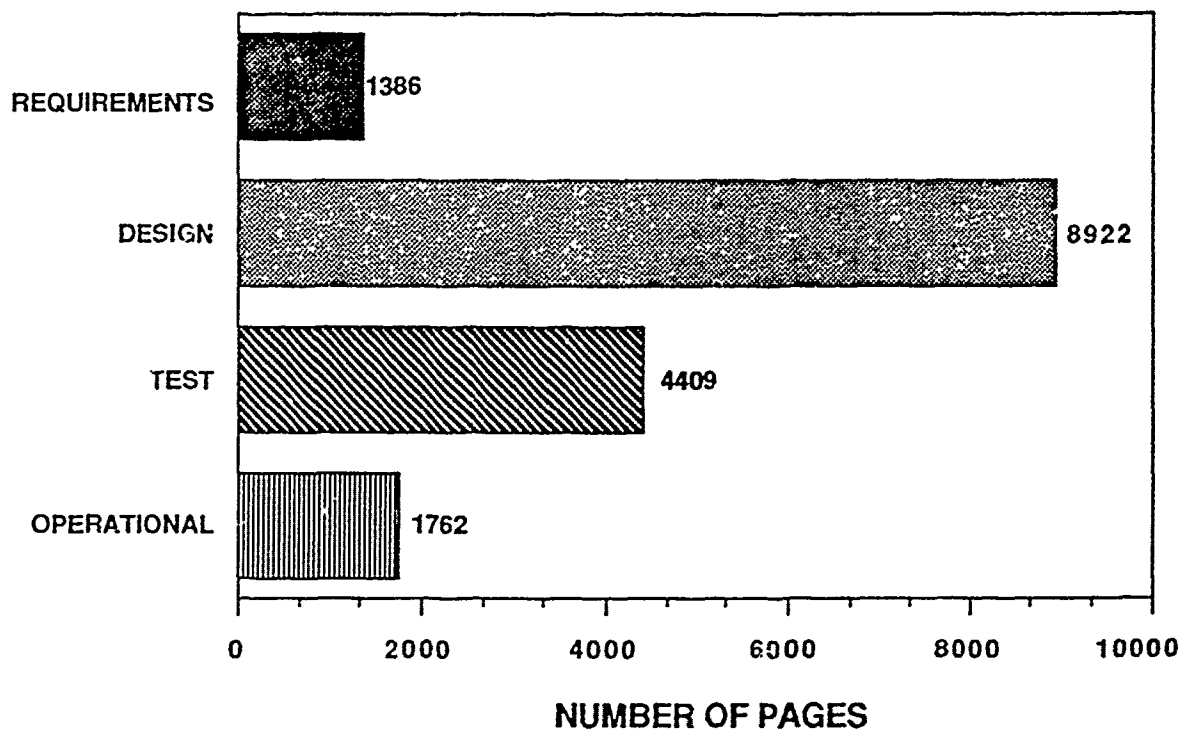


FIGURE 6.2-1 RELATIVE DOCUMENT SIZES BY CATEGORY

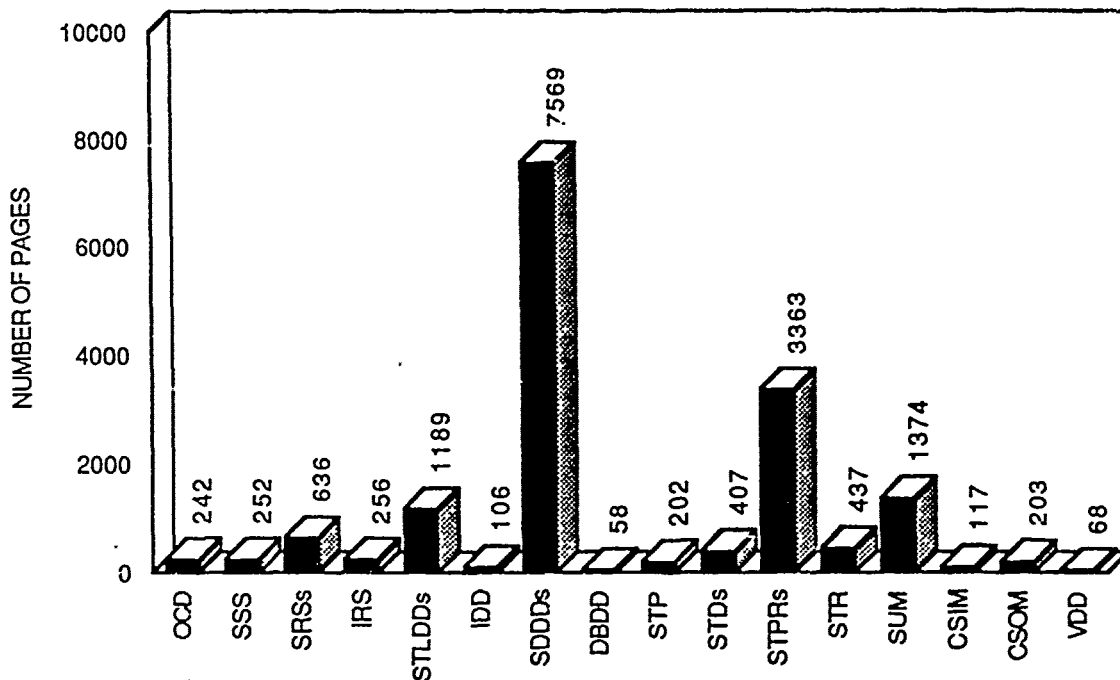


FIGURE 6.2-2 RELATIVE DOCUMENT SIZES BY DOCUMENT

6.3 Deliverable Source Code

In order to create an operational TISSS system, various types of source code were necessary. The most obvious was the Ada source code which formed the executable programs which made up TISSS. There was 112,918 lines of code of Ada. (Note : A line of code is considered a physical line excluding blank lines and comment lines.) There was also some use of VMS operating system features which required 4,453 lines of DEC Command Language (DCL) to be delivered with the system. DCL (4,453) represents a small percentage (3.7%) of the total source (119,444) in the attempt to minimize the dependency of the operational system on the host environment. In order to use the Oracle data base, 952 lines of Structured Query Language (SQL) was written. Finally in order to define the user interface menus and mechanisms to get the user input data into the software, 22,391 lines of TDMS source was written along with 10,359 of menu help text. (Reference Figures 6.3-1, 6.3-2, 6.3-3 and 6.3.4 for details).

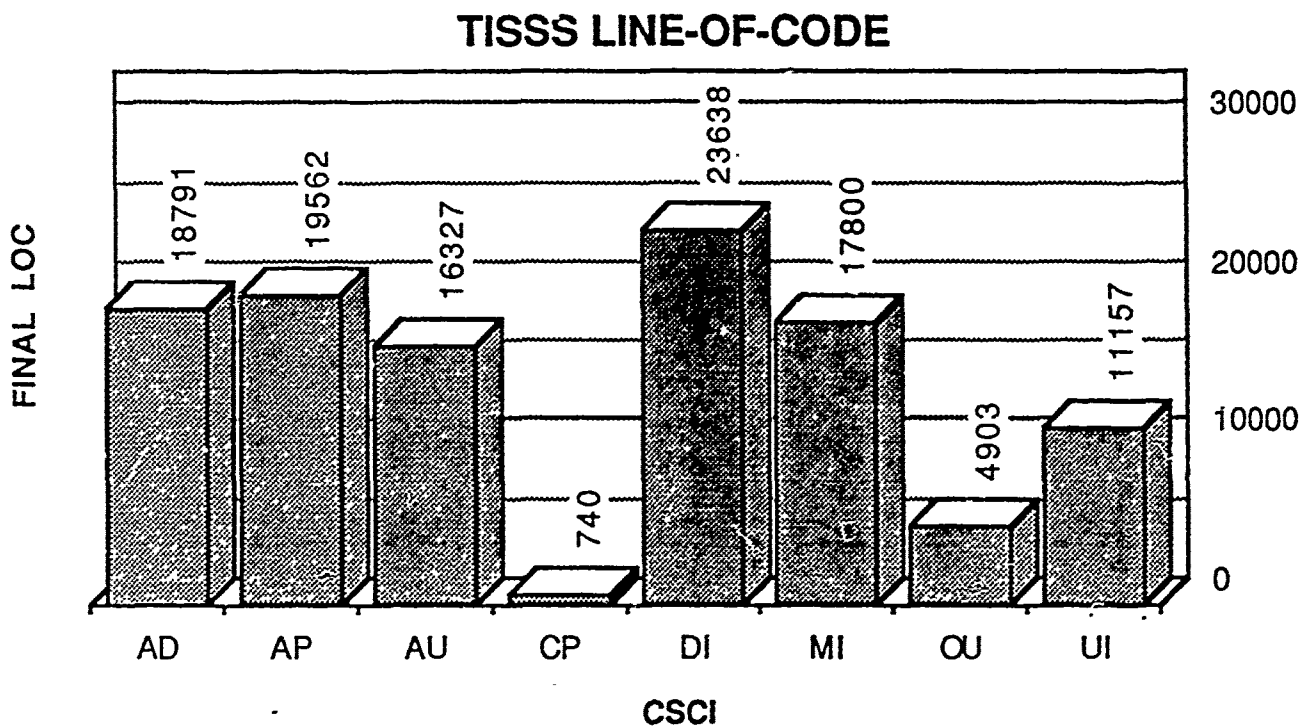


FIGURE 6.3-1 TISSS LINES OF CODE BY CSCI

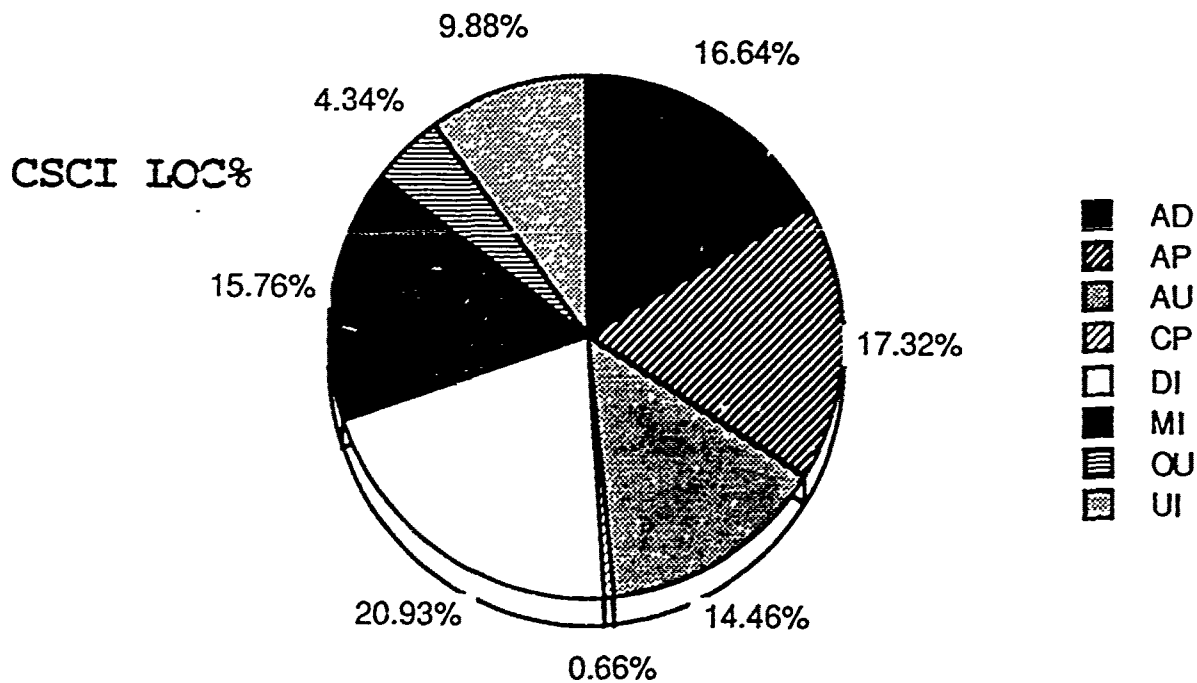


FIGURE 6.3-2 TISSS LOC BY CSCI PERCENT

	<u>SOURCE</u>			<u>TEST</u>			<u>FORMAL TEST</u>		
	<u>ADA</u>	<u>COM</u>	<u>SQL</u>	<u>ADA</u>	<u>COM</u>	<u>SQL</u>	<u>ADA</u>	<u>COM</u>	<u>SQL</u>
AD	18791	1047	273	26038	49661	1538	1990	11395	519
AP	19562	0	0	12535	1018	0	15	8740	0
AU	16327	0	0	6892	2070	0	2334	6494	0
CP	740	0	0	848	300	0	14	1296	0
DI	23638	0	0	22176	2054	0	2033	5277	0
MI	17800	0	0	23221	1441	0	1138	17187	0
OU	4903	0	0	2997	675	0	641	2460	0
UI	11157	456	0	3895	347	0	1466	1243	0
SYSTEM	0	2950	679	0	1108	0	0	2285	46
TOTAL	112,918	4453	952	98,602	59,174	1538	9,631	56407	565

FIGURE 6.3-3 TISSS LOC - SOURCE VS TEST

	<u>LOC</u>
TDMS HELP	10359
TDMS RECORDS	1786
TDMS REQUESTS	4910
MENUS	15695
	<hr/>
	32750

FIGURE 6.3-4 TISSS LOC - TDMS AND MENU FILES

6.4 Test Code

In order to test a software system to the level required for field operation, many software drivers and stubs had to be written in order to perform unit testing, package testing, TLCSC integration and finally CSCI testing at the formal level. Informal testing which is done at the CSCI level and below caused 98,602 lines of Ada code to be written for this purpose. Formal test which was performed at a CSCI level required 9,631 lines of Ada code to be developed to isolate the CSCI boundaries. So it can be observed that 108,233 LOC were written to test 114,039 LOC or 94.9%.

The testing activity was automated as much as possible by using DCL command files for setting up the data and the environment, compiling, linking and simulating user input. 55,837 LOC DCL command files were written for informal test and 56,407 LOC DCL command files were written for formal test. This total, 112,244 compared to 108,233 LOC Ada (103.7 %) shows a high degree of automation. By referencing Figure 6.3-2, it is also seen that 2,103 LOC SQL was written in the test effort.

6.5 Commenting

Figures 6.5-1, 6.5-2 and 6.5-3 shows the comparison of lines of comments associated with the lines of code for each CSCI for deliverable source, informal test and formal test code respectively. The amount of commenting corresponds appropriately to the visibility and usage of each type of code. Deliverable source code which will be maintained with the system throughout its lifetime should be well documented and explained. This is shown by 180,930 line of comments for 114,039 lines of Ada or 158.6%.

<u>CSCI</u>	<u>LOC</u>	<u>COMMENTS</u>
AD	17644	38921
AP	19562	28138
AU	16327	29534
CP	740	1120
DI	23638	32370
MI	17800	28142
OU	4903	6918
UI	11157	15309
SYSTEM (AD)	1147	478
TOTAL	<hr/> 112,918	<hr/> 180,930

FIGURE 6.5-1 TISSS LINES OF CODE - SOURCE ADA

<u>CSCI</u>	<u>LOC</u>	<u>COMMENTS</u>
AD	26038	9444
AP	12535	9040
AU	6692	1356
CP	848	1005
DI	22176	4312
MI	23221	8135
OU	2997	2450
UI	3895	2270
TOTAL	<hr/> 98,602	<hr/> 38,012

FIGURE 6.5-2 TISSS LOC INFORMAL TEST SOFTWARE

<u>CSCI</u>	<u>LOC</u>	<u>COMMENTS</u>
AD	1990	653
AP	15	1
AU	2334	401
CP	14	0
DI	2033	2534
MI	1138	1678
OU	641	754
UI	1466	1262
	<hr/>	<hr/>
TOTAL	9,631	7,283

FIGURE 6.5-3 TISSS LOC - FORMAL TEST SOFTWARE

7 PUBLICATIONS PRODUCED

The following publications and presentation were produced as a result of activities performed on the TISSS project.

1. Ardito, M. and Donnellis, G., "Using ORACLE to Implement a Hybrid Data Base", ORACLE International Users Group Conference, Washington, D. C., September, 1987
2. Freeberg, T., "Ada and DOD-STD-2167: Experience on a 100K Line of Code Project", AIAA Computers in Aerospace Conference, October 1987.
3. McCoy, L. S., "Interfacing Ada and Relational Databases", Ada LETTERS, ACM SIGAda, Volume VII, Number 3, May/June 1987
4. McCoy, L. S., "Interfacing Ada and Relational Databases", Proceedings of the 1987 Oracle International User Week
5. Rolfe, R. M., Tester Independent Support Software System, Panel Session, International Test Conference, Washington, D.C., Sept. 1988.
6. Rolfe, R. M., Tester Independent Support Software System, for MIL- M-38510 Microcircuit Qualification, VHSIC Qualification Conference, Colorado Springs, Aug. 1987.
7. Rolfe, R. M., Lehtonen, D., et al, TISSS in the CALS (Computer Aided Logistics- Support Environment, Annual Reliability and Maintainability Symposium, Philadelphia, Jan, 1987.
8. Rolfe, R. M., Tester Independent Support Software System, International Conference on ATE, Best of International Test Conference Session, invited Paper, Paris, September 1986.
9. Rolfe, R. M., Tester Independent Support Software System, a User's View, VHSIC Qualification Workshop, Vail, Sept. 1986.
10. Rolfe, R. M., Hardware Independent Test Data Standards, Proceedings Computer Standards Conference, San Francisco, May 1986.
11. Rolfe, R. M., Tester Independent Support Software System, Proceedings of IEEE Test Conference, Philadelphia, Oct, 1985. (selected in best five papers of conference).

8 GOVERNMENT PATENT

A patent proposal covering the design concepts of the TISSS AMTE Postprocessor was submitted to Rome Air Development Center (RADC). A copy of the patent proposal is given in Appendix D. RADC will submit a patent application.

9 TISSS INSERTION RECOMMENDATIONS

TISSS has been built, now it must be used. The following paragraphs cite recommendations or disclose plans to ease acceptance by the community of potential users.

9.1 Upgrade To Current Versions Of Embedded Software

The developed TISSS runs under on VAX VMS 4.5, which has been superseded by VMS 5.0. The TISSS is planned to be upgraded to run under the new operating system. This implies an upgrade of all embedded COTS tools. These include DEC software, specifically TDMS and Runoff, and third-party software, including Oracle, Palette, HILO, and HITS (note that HITS is owned by the Navy).

9.2 Upgrade To IEEE 1076

TISSS currently uses VHDL 7.2 for models and test vectors. The TISSS should be upgraded to the standard version of VHDL, IEEE 1076. Thereafter, the modeling subsystem of the TISSS should be upgraded to IEEE 1076 compatibility.

9.3 Upgrade TVL

The TISSS Test Vector Language (TVL) is planned to be upgraded from VHDL 7.2 to IEEE 1076 as part of the previous recommendation. In addition, numerous enhancements have been identified that will make TVL acceptable to a wider range of users and will therefore expedite the process of converting TVL into an industry standard. These enhancements are described under the Future Directions section.

9.4 Additional Test Philosophies

TISSS was delivered with a test philosophy for LSTTL technology microcircuits. Additional test philosophies should be developed, targeted for VHSIC-level technologies. At a minimum, a CMOS test philosophy would be required. New test macros may also be defined, to augment the current set. By providing test philosophies, standards and conventions are established, and new TISSS sites can immediately begin generating test specifications.

9.5 Beta Sites

A number of TISSS beta sites have been identified. Five government locations have been selected as beta sites, DESC, SA-ALC, SM-ALC, WR-ALC, and JPL. In addition, a number of potential industry beta sites within the US have volunteered, although final selections have not been made. There is also interest from foreign industry, especially from the French embassy. However TISSS is governed by ITAR restrictions so the code cannot be shipped to overseas users as yet.

The beta site effort is scheduled to last eight months, followed

by a three month evaluation period. Beta sites will receive training and technical support. Problems identified during this period will be considered by the TISSS Configuration Control Board, which will evaluate the problems and prioritize suggested enhancements and corrections.

9.6 Data Availability

To facilitate acceptance of TISSS and the development of more tools, complete documentation sets are planned to be made available by the Government. In addition, the source code for the developed CAD Postprocessor and AMTE Postprocessor will be made available. The source code for the other TISSS subsystems will generally be unavailable to prevent the development of nonstandard TISSS systems.

A number of tools were developed under the TISSS program. These tools will be made available as well, with source code and limited documentation.

9.7 Postprocessors

The development of additional AMTE Postprocessors is a very important part of the TISSS insertion process. Without an AMTE Postprocessor, a TISSS site can generate MIL-Specs but cannot generate a test program to actually test devices. Such a TISSS site will be unable to realize the cost savings of automatic test program generation. TISSS currently provides an AMTE Postprocessor for the GenRad GR-18. Two additional postprocessors are planned, one of which will be developed by RADDC and one which will be openly procured. These will be targeted towards AMTE/ATE in use by the Government.

9.8 DLA/SPO Coordination

TISSS has received the endorsement of the CALS program office and has been included in MIL-STD-454, Requirement 64, dated 22 September 1988. TISSS is being evaluated for the ATF program, and a board level extension to TISSS, called LRM TISSS, has received strong support from the ATF program. TISSS is also being used for the GVSC program, contractors, meeting the requirement that test information be provided to RADDC in TISSS compatible format.

9.9 Insertion Issues From The TISSS Industry Review

A number of issues were raised at the TISSS Industry Review held June 22,23 1988. These issues become action items to be addressed by the Government insertion plan.

9.9.1 How To Achieve Effective TISSS Buy-in By Merchant IC Houses

How can the government effectively get merchant semiconductor buy-in to the use of the TISSS and its input formats?

Initial Response: Most merchant IC vendors are very concerned about disclosure of detailed gate level models of their standard product to the Government or any other customer. These standard products represent major R & D investment for these vendors. A new microprocessor development can exceed \$50,000,000. With this kind of investment in intellectual property it is easily understood why there is concern. The entire investment can be transported on one or more TIF tapes as depicted in Figure 9.9.1, the Complete TISSS Input Format (TIF). Various users need subsets of information from the complete TIF. We will describe some of the necessary data transfer subsets in this response.

9.9.2 TISSS Usage In Plant

Major merchant IC vendors at the TISSS Industry Review Meeting indicated that they would be very willing to host TISSS at their plants for those products to be qualified for DoD usage. They would distribute the Test Only TISSS Output Format (TOF) data shown in Figure 9.9.2 to meet the data needs of the OEM and Government for product support. The data sets depicted in Figure 9.9.2 with shaded fill are not provided on the TOF. This Test Only TOF is created by appropriate TIF/TOF flag setting for each data set associated with the device/version stored in the TISSS data base. All logical models, schematic, layout, total fault dictionary, and device interface SRUs would be marked TIF only.

9.9.3 Standard Product Integrated Circuits (IC)

Merchant semiconductor vendors will want to provide the OEM user of their standard components adequate information to reliably use their components during OEM product development. In addition to the Test Only TOF, additional information found in the Design Requirements TIF is necessary to support many system applications. This necessary information has been indicated to be product and test specification data and adequate simulatable models to allow integration of these standard components into custom OEM equipment. The minimum information for this data transfer is described in Figure 9.9.3, the Design Requirements TISSS Input Format (TIF).

The Government Qualifying activity or representative would come to the merchant semiconductor plant to audit the complete TISSS microcircuit description. The merchant semiconductor would retain in house the Complete TIF as depicted in Figure 9.9.1 for standard products.

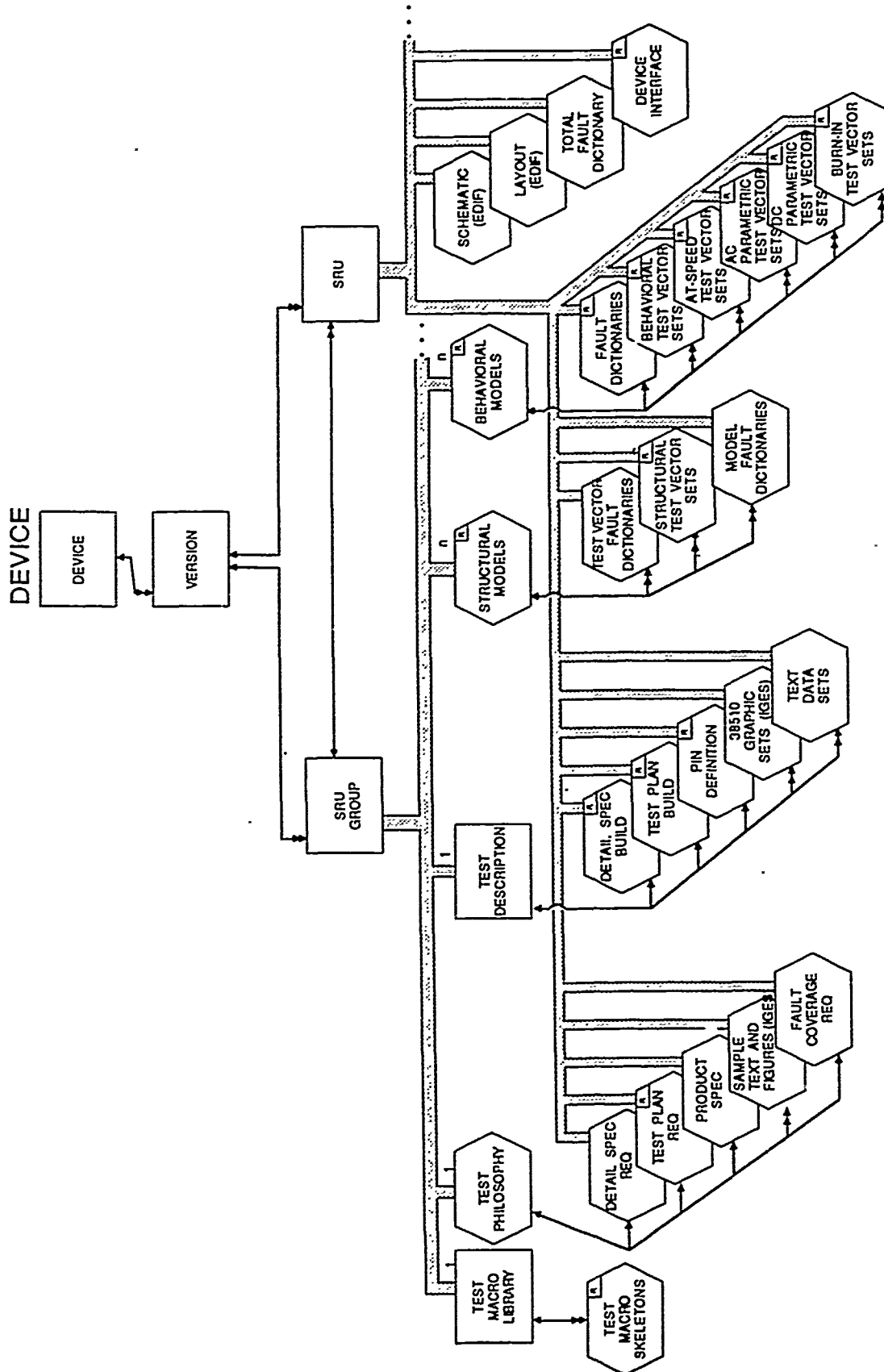


FIGURE 9.9.1 COMPLETE TISS INPUT FORMAT (TIF)

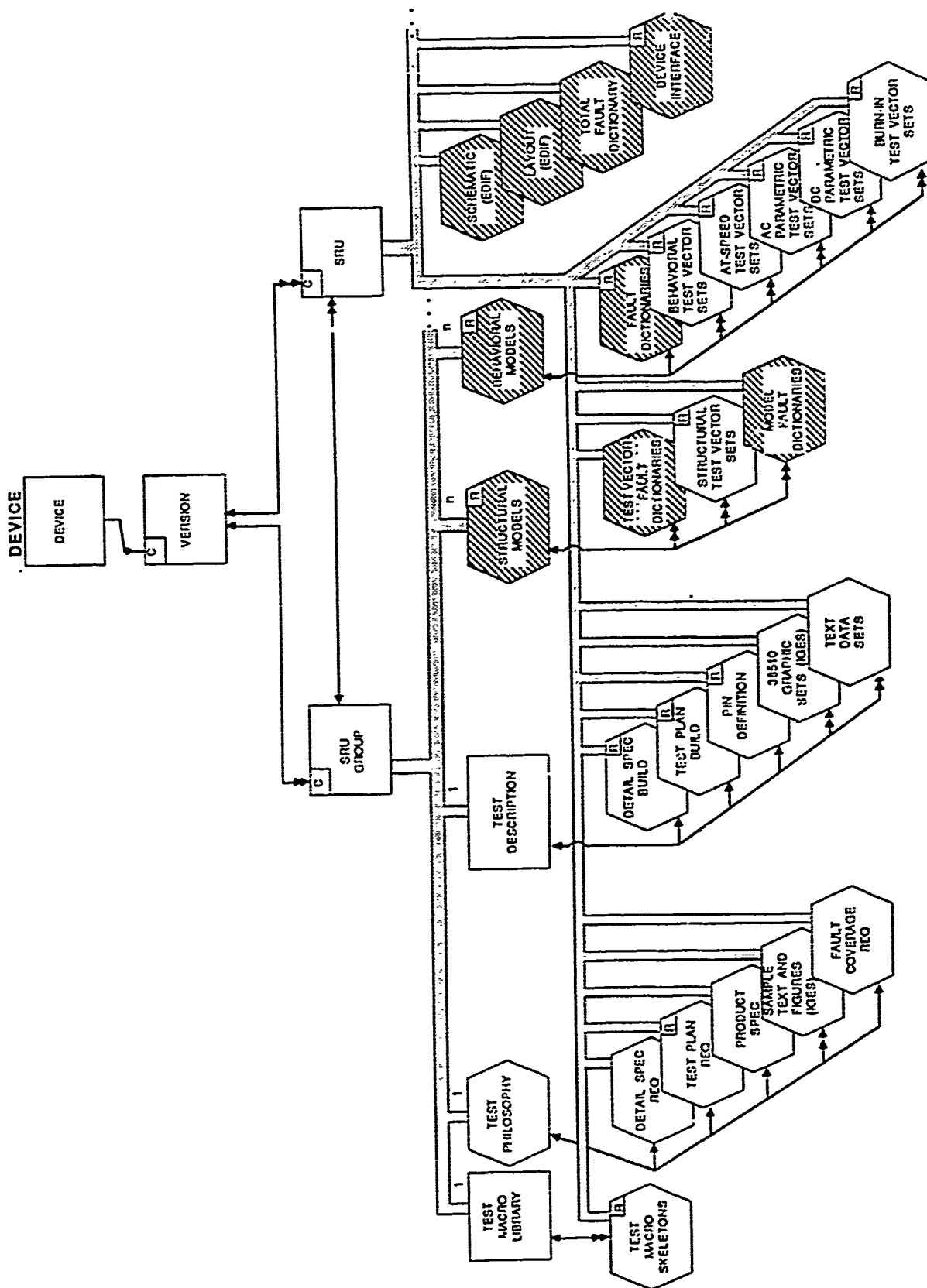


FIGURE 9.9.2 TEST ONLY TISS OUTPUT FORMAT (TOF)

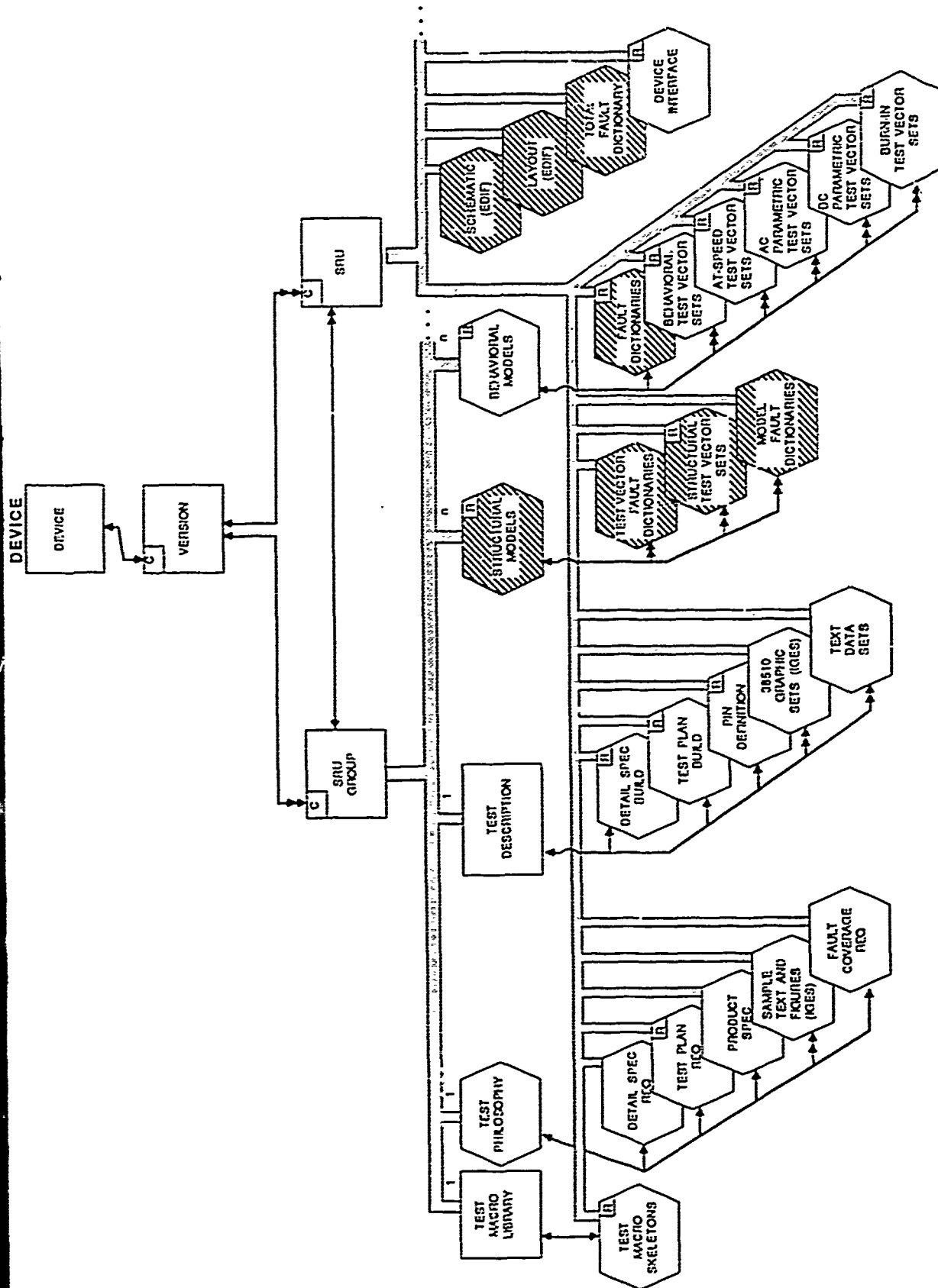


FIGURE 9.9.3 DESIGN REQUIREMENTS TIF

9.9.4 Application Specific Integrated Circuits (ASIC)

For application specific products of any type, the minimum TIF to be provided to the semiconductor manufacture by the OEM product developer is depicted in Figure 9.9.4, the Design Requirements TIF. For this particular TIF type, the shaded items are those items not required to specify a design requirement. The shaded items include structural information consisting of schematic, layout, total fault dictionary, and all models except required behavior models. Of course all of the portable test vectors sets are provided for the required behavior.

9.9.5 Form, Fit, And Function And BIT Or BIST

Structural test vector sets are not necessarily part of the design requirements for an ASIC product. This is a very important issue for some kinds of built in test (BIT) or built in self test (BIST) microcircuit devices. Form, fit and function replacement circuits may require the same BIT or BIST response. Unfortunately, BIT and BIST responses often includes tests of device structural integrity. Advanced VHSIC Phase II circuitry might include an "I'm Okay signature" on the Element Test and Maintenance Bus (ETM) which is independent of structure used to implement a device to a specific design requirement. In any case, if structural BIT or BIST signatures are part of the design requirement, the behavioral test vector sets should verify the BIT or BIST design requirements. Further, the behavioral models must be based upon the required structural architecture to provide the correct BIT or BIST structural signatures or response.

9.9.6 A Special Case For ASICs

There are some ASIC devices (e.g. simple gate arrays) which may be remanufacturable without the layout description. However, these devices must be based on de-facto standard cells and layout rules. Unfortunately, these standard cells and layout rules change typically within a few years.

In this case, only the data from the design requirements TIF will be necessary for archive, because remanufacturing will almost always include some redesign activity (minimally design verification), even if highly automated. If the gate level design is part of the design requirement, then a gate level model should be provided as one of the behavioral models.

9.9.7 Where Is The Complete TIF ?

The manufacturer of the ASIC circuit will maintain the complete TIF description for each component. This TIF description would be available to the Government Qualifying activity or representative to audit compliance with contract data storage requirements for TISSS microcircuit descriptions.

It is advisable to have a complete TIF description including manufacturing process identification archived somewhere. However, this archive may depend on the contractual agreements. For individually qualified microcircuits which are proprietary, the data can be stored with the qualifying vendor. However, contractual protection will be required to assure availability over the long military life cycles.

9.9.8 Update SUM Version Information For COTS

Update the Software User's Manual to provide sufficient information to allow a user to order Commercial of the Shelf Software (COTS) necessary for the TISSS operation without technical support.

Initial Response: As part of the Beta site support activity, the SUM should be updated as necessary with a better description of the COTS within the TISSS. For each COTS item the following should be provided:

- a. The COTS manufacturer's product name for ordering purposes.
- b. A brief product description.
- c. A description of the product's application to TISSS functions and missions.

The Version Description Document (VDD) should be updated to provide the current COTS product description and version number necessary for procurement for each TISSS release.

It is anticipated that future TISSS releases will incorporate the latest release of COTS products to maintain the high quality level of support that the TISSS user will expect.

9.9.9 Update SUM Hardware Description Information For Laser Printers

SUM hardware description does not include a DEC LN03 laser printer for specification generation.

As part of the Beta site support activity, update the SUM minimum hardware requirements to reflect hardware requirements for

hosting the complete TISSS.

9.10 TISSS Test Philosophy Minimum Test Set/Additional Tests

The customer has raised the following concern as one that the TISSS system should accommodate. "Generic test requirements can be defined as the minimum set of requirements that must be performed on a particular device class/technology. The ability to add additional tests for an individual device directly into the TEST PLAN.BLD from the available test macro skeletons must exist. This will allow a TISSS user the ability to add additional tests that may be appropriate for an individual device and not required by other devices in the same associated class/technology. This assumes that the device is appropriate for the class/technology and does not constitute the creation of a new class/technology."

The TISSS system as implemented in version 1.0 does fully allow the scenario described above. The Test Philosophy (also known as pseudo device) identified by a class and technology in the TISSS data base provides the minimum set of test requirements that must be performed on all devices of a particular class/technology. When a device of that class/technology is created the pseudo device Test Philosophy is copied as a SRU group part of the newly created device/version. (Reference section VII.3.3 of the TISSS Software User's Manual on TISSS Conceptual Models). The Test Plan Build SRU that is part of a device/version is only checked against the Test Plan Requirement that is in the Test Philosophy group that is also in the same device/version. It is not checked against the Test Philosophy pseudo device.

When additional tests are appropriate for an individual device but not required by other devices in the same class/technology, a new version of the pseudo device should be created using the 9.4.1 Copy Test Philosophy menu. It is recommended that the version name (4 characters) be an abbreviation of the device requiring the variation of the minimal Test Philosophy. A suggested version name for the minimal Test Philosophy pseudo device is BASE.

The additional tests may then be added to the new version of the pseudo device using the 9.3 Modify Test Philosophy SRUs menu. Then the revised test philosophy can be copied into the Test Philosophy group of the device requiring additional tests by using the 2.4 Copy SRU Group menu. The auditing functions will ensure that the test plan developed for this device includes all additional tests.

When a device is submitted for qualification, the associated Test Philosophy SRUs should be extracted from the device description and compared using a basic Difference tool to the approved Test Philosophy for the Class/Technology. If the differences indicate additional tests were added but no changes were made to the minimum test philosophy and if these additional tests are acceptable to the qualifying agency, the device should be

approved.

By use of the Test Philosophy library features of TISSS, the user can add augmented test philosophies for each device and provide a special library version as needed. Each device carries its own unique test philosophy. All interactive and batch audits are based on the device test philosophy. Specifically modified test philosophy versions may be compared to the minimal base version.

10 CONCLUSIONS AND RECOMMENDATIONS - TISSS FUTURE DIRECTION

10.1 Modeling - Hardware Accelerator

One area of TISSS that needs expansion is the Modeling Subsystem. Currently no true value simulation capability is available. The fault simulation and ATPG capabilities are limited by the simulation environments integrated into TISSS. The question then has to be asked of why fault simulators were incorporated?

The purpose was to prove a device met fault coverage requirements. This may be easy enough to prove with the embedded TISSS simulators if all device were basic combinational logic. This is never the case. Design for testability techniques and built in test structures make proving fault grade more complicated.

The approach of fault simulating within TISSS is only feasible for certain types of integrated circuits. The translation process that must occur from VHDL to another modeling language limits the true representation of components on the IC. In hindsight, concentration should of been focused on an audit procedure that accepted fault simulation reports. These reports would include fault detection data as well as built-in-test signature analysis. The audit procedure would verify that all nodes were tested and calculate an overall fault coverage for the entire device. Any further validation of fault coverage would be performed on site at the IC manufacturer.

True value simulation is a good way to prove design intent. The VHDL simulator should become a part of TISSS. It could be integrated or accessed through files in and out of TISSS. Due to the complexity of VHSIC and the future requirements for LRMS, a hardware accelerator should be evaluated.

There is currently a task under TISSS Insertion to evaluate the ZYCAD Mach 1000. Input to the accelerator is VHDL structural representations. Any VHDL limitation needs to be evaluated and documented. The simulation report function will be examined to determine pertinent information that could be extracted for TISSS.

The future of Modeling depends heavily on further development with VHDL. Work needs to progress to develop more components in the TISSS standard VHDL library. A basic set of primitives was defined for TISSS that could easily map between HITS and HILO-2. This basic set needs to be enhanced as more and more VHDL designs become available. Modeling techniques need to be explored for expressing bottom up and global timing, analog representations, and technology specifications at a primitive level. A library of acceptable models should be maintained. As VHDL becomes more prevalent, higher level functions should be included. Ideally, the library should grow to contain descriptions of behavioral representations of commonly implemented and also complex

functions of VLSI components. This includes design for testability techniques and built-in-test structures.

Through TISSS, Harris was able to develop and define interfaces and architectural body formats. More importantly, interfaces to the VHDL test bench required for simulation were defined. One major accomplishment was the signal generator whose definition provided an interface for TVL, the TISSS test vector language. The effort of standardizing VHDL interfaces to the test bench need to continue. Once a standard for TVL is established a standard signal generator could be developed. This would help users in the VHDL environment understand TVL and aid them in the simulation process.

For the future, it is important for the VHDL simulator to become a part of TISSS. Either fully integrated or accessed through files in and out of TISSS. The current VHDL library needs to be expanded to express multiple functions including design for testability techniques and built-in-test. A hardware accelerator would make simulation much more feasible on large circuits. Standards in fault simulation report formats need to commence. Each step aids in the progress of developing a unified VHDL design and test environment.

10.2 Test Specification (TDL) And Description (TVL)

The interface to the TISSS system is defined by the TISSS Input Format TIF. A number of languages are defined within TIF, including the Test Description Language (TDL) and the Test Vector Language (TVL). TVL will be upgraded as part of the TISSS insertion plan. TDL may be upgraded as part of the LRM TISSS program. This section briefly describes some desirable enhancements to these languages.

10.2.1 Test Description Language (TDL)

The Test Description Language (TDL) is used to express test requirements and test specifications. Extensions include language standardization, syntax extensions, additional semantic rules, and conventions for language use.

10.2.1.1 Language Standardization

TDL was developed as part of the TISSS program, and conforms to no existing industry standard, although it resembles Ada and VHDL. It would be beneficial to either migrate TDL to an existing standard, such as IEEE 1076 (VHDL) or EDIF, or to promote TDL as a standard in itself. In the process of standardization it will be important to avoid making the language too complex. A simpler representation that still meets the objectives of the language will be more easily supported by new tools.

10.2.1.2 Additional Test Macro Types

The following changes to the pre-defined test macro types are recommended.

- a. Eliminate inconsistencies between the pre-defined TDL types used in the test plan requirement and the corresponding types used in the test plan build. In some cases the field names of records do not agree, in other cases, the requirement type does not provide enough defaults for the build type.
- b. Express the accuracy (allowable error) of a physical value with a percentage and a minimum, rather than as a single value. This supports meaningful defaults for accuracies as well as audits of accuracies.
- c. Provide array parameters in test macros, which allow a test engineer to specify a list of values, rather than a single value.
- d. Alter TDL in response to modifications of the Test Vector Language (TVL). For example, if TVL supports labels, then TDL should allow vectors to be referenced by label as well as by vector number.
- e. Eventually, TISSS should be extended to handle hybrid and analog devices in addition to digital devices. Such devices can be handled within TDL with the addition of new parameter types. For example, a parameter type might be required that described a tone burst for communications devices.

10.2.1.3 Additional Semantic Rules

Certain portions of a test description are currently validated manually. Additional rules should be defined to further automate the audit process. Some rules will require additional TDL structures while some will only require changes to the language semantics and tools.

- a. Rules dealing with the specification and use of the setup conditions table should be expanded.
- b. Rules dependent on the interaction between test macro parameters (eg, pinset A must be a subset of pinset B) should be implemented.
- c. An extensible rule system should be investigated, in which a newly developed test macro may contain previously undefined rules, perhaps expressed in a language such as Prolog or LISP.

All new rules should be evaluated to ensure that they are not

overly restrictive.

10.2.1.4 Object Names

A consistent naming procedure should be established that allows a direct reference into test description tables. This naming procedure would be used consistently by all TISSS tools, including report generators, AMTE Postprocessors, and design input tools. The naming procedure should take advantage of the hierarchy of TDL.

10.2.1.5 Audit Exceptions

An automated exception acceptance technique could be implemented which records individual errors detected by an audit of a test specification, and suppresses those errors from further audit reports. Along with the errors to be ignored, the database would record the auditor authorizing the exception, the date authorized, and other configuration management data. With respect to TDL, this implies a standardization of error reporting and a means of associating errors with individual TDL objects.

10.2.2 Detail Specification Requirement/Build

Although not a formal part of TDL, the languages used to express the Detail Specification Requirement (DSR) and the Detail Specification Build (DSB) could be enhanced to take advantage of advanced document production technology. TISSS currently uses DEC Runoff for document production. An upgrade to more sophisticated document production software would support variable width fonts, subscripts and superscripts, automatic figure and table numbering, and automatic figure insertion. A change from DEC Runoff would also eliminate any potential difficulty that might arise when migrating TISSS to a different operating system, such as Unix.

If more sophisticated typesetting capability is provided, procedures for using, say, subscripts and superscripts within test macro symbols should be established. This will allow tables to be generated that more closely resemble current formats.

The syntax of the DSB could be simplified to reduce the cost of support tools. Restrictions on SRU references could be removed, allowing a test engineer to specify text or graphics information for the Detail Specification as desired.

A powerful tool would be the provision of user-definable table formats. This capability could be provided with the same technology used in the TISSS AMTE Postprocessor. It would allow individual sites to produce tailored reports for internal use, and official MIL-Spec reports for government use. A related capability would be product specifications that are independent

of the MIL-M-38510.

10.2.3 Test Vector Language (TVL)

Plans exist to upgrade and enhance the current TVL. The intent of these enhancements is to reduce the storage size of test vector sets, to support structured (hierarchical) test vector sets, and to support Built-In Test (BIT) and Built-In Self Test (BIST). Additional future directions include a library of standard hardware interfaces and support for analog test.

Existing plans call for three levels of TVL. The first level is a simple representation with features that allow data compaction. This TVL will be easily processed to generate a flat vector set. The second level supports hierarchical test vector sets. Procedures would be defined that could be called from the main vector set. However the vector set could still be expanded to a flat level-1 vector set. The third level includes conditional vector sets (for dealing with asynchronous outputs) and the ability to use VHDL behavioral models to generate test vectors. In general, a level-3 TVL will require a VHDL simulator for expansion, and the vector set will not necessarily be translatable to a level-2 or level-1 representation.

A hierarchical TVL, as currently envisioned, will support standard protocols. If a device is to interface with a PI bus, for example, a test vector set will have a procedure that takes a binary word as input and generates the necessary set of vectors required to write that word on the bus. A similar procedure would be written to read a word from the bus. Eventually, standard procedures will be defined and collected in a library. This simplifies vector sets while eliminating the possibility of error in specifying the interface.

To support analog testing, digital test vectors may have to be coordinated with analog signals such as voltage ramps or tone bursts. This coordination could be handled from within the vector set. For example, each test vector could contain the voltage required for the ramp, or two frequency values specifying the two tones in a tone burst.

10.3 Component And Board Level

The next logical step for TISSS is to extend the concept to the board and LRM level. This extension requires additional information stored in the database as well as additional database relationships.

Additional data required includes physical information, diagnostic information, and embedded software. Physical information is needed to describe the dimensions of the LRM and locations of connectors and internal circuit nodes. Diagnostic information is needed to locate faults within the LRM, for ultimate repair. (A major difference between components and LRMs

is that LRMs are repairable while components are not.) Diagnostic information would include guided probe data as well as information about BIT/BIST within the module. Also, since many LRMs have embedded software which affects the functionality of the LRM, this software must also be stored in the database.

The database must be extended to support the sometimes complex revision and version status of LRMs. For instance, two LRMs might differ only in the embedded software, and this distinction must be captured. The database must also allow reuse of previously entered data, so that LRM test specifications can make use of component test specifications.

10.4 Database And Tool Extensions

The most important part of TISSS is the standard TIF interface. Any organization that can generate a TIF supports TISSS, whether or not that organization uses any developed TISSS tools. However, most organizations that wish to use TISSS will first look to existing tools. This section lists enhancements that would make the TISSS database a more attractive tool.

- a. Rehost to Unix.
- b. Provide a network interface to the database.
- c. Provide a secure interface to the database at the operating system level, to allow tools to be built from command procedures.
- d. Support compression of data stored in the database, especially for large vector sets.
- e. Provide separate read/write access for each SRU in the database, based on user ID or membership within some group.
- f. Allow a test description to be audited against a test philosophy in a pseudodevice.
- g. Support common directory/file operations such as delete and rename for devices, versions, and SRUs.
- h. Support default SRU classes, names, and attributes.
- i. Establish procedures and conventions for storing tester specific information within the TISSS database. Example tester specific data include fixture configuration files and test results (datalog output).

APPENDIX A
FINAL TISSS STATEMENT OF WORK

TISSS SOW Task (4.1)

(Phase I) The contractor shall:

TISSS SOW Task (4.1.1)

Review, as a minimum, the work described in Appendix 1, "Pertinent Research Topics," and shall summarize each and report on its applicability to a TISSS (see CDRL).

TISSS SOW Task (4.1.2)

Investigate the possibility of using existing software packages and test languages so that the cost of the system may be minimized. As a minimum, evaluate the software packages and language listed in Appendix 2, "Available Software packages," and report on their status, availability, ownership, cost, operation, and potential for use in the TISSS (see CDRL).

TISSS SOW Task (4.1.3)

Design a TISSS, using the results of the tasks described in paragraphs 4.1.1 and 4.1.2 and using Appendix 3, TISSS System Description and Design Goal", as information and as a design goal. The contractor shall advise the contracting officer of any deviations from Appendix 3. The TISSS design and program plan shall be in accordance with Attachment No. 1. The contractor shall consider suggestions made by the Government at the Preliminary Specification Review (PSR), see paragraph 4.3.2.

TISSS SOW Task (4.1.4)

Evaluate hardware selection and software language selection during the design phase. Evaluate such factors as user friendliness, the desirability of software transportability, the types and specifications of the host computers required for the development and operation tasks, and the selection of higher order languages such as Pascal or Ada, as well as their run-time libraries, including restricted subsets of languages that would be truly machine independent. Evaluate the feasibility of using only the features of a language that are supported in Ada so that a conversion to Ada would be possible in the future. Evaluate the Government's computing

capabilities as described in Appendix 4, "Computing Facilities," to determine any potential host capabilities for system development and operation. Submit an analysis of the various hardware and software alternatives to the government and discuss the costs associated with each alternative (see CDRL).

TISSS SOW Task (4.1.5)

Evaluate the acceptability of the designed system to the manufacturers and users of MIL-Spec microelectronics. Investigate whether manufacturers will willingly release proprietary gate and layout information to the Government (assuming it will be protected) and what possible TISSS design modifications or Government actions might facilitate this. Analyze the impact of this system so that a meaningful and standard is promulgated (see CDRL).

TISSS SOW Task (4.1.6)

Perform a preliminary design in accordance with Attachment No. 1, (see CDRL).

TISSS SOW Task (4.1.7)

Conduct a technical presentation on his design at a conference on the TISSS program at a time and place, as specified in the Contract Schedule. This presentation shall contain a general introduction to the contractor's system and specific information on the tester independent attributes of the system. This conference will be held for the purpose of soliciting comments from Government and industry representatives on each contractor's design.

TISSS SOW Task (4.1.8)

Prepare a detailed technical and cost proposal for the implementation of this design (Phase II) (see CDRL)

TISSS SOW Task (4.2)

(Phase II) The contractor shall develop, implement, and validate his designed TISSS system and shall deliver it to the Government. This system shall be in full accordance with the Requirements Baseline (Attachment No. 4), except that the VHDL Simulator shall not be required. The VHSIC data prep and beta testing requirements set forth in paragraphs 4.2.15 and 4.2.18 will also not be required. VHDL product evaluation tasks shall be performed. Within the Requirements Baseline the following order of precedence shall apply: System/Segment Specification, Hardware Configuration Baseline, Software Standards and Procedures Manual, Software Development Plan, Software Quality Assurance Plan, and the Software Configuration Management Plan. All software developed under this task shall be in full accordance with Attachment No. 1 (DoD-STD-SDS). The contractor shall:

TISSS SOW Task (4.2.1)

Perform a preliminary design in accordance with Attachment No. 1,

(DOD-STD-SDS), Section 5.2, including 5.2.1.5 and 5.2.1.6 (see CDRL).

TISSS SOW Task (4.2.2)

Conduct a detailed design in accordance with Attachment No. 1, Section 5.3 with the exception of paragraphs 5.3.1.11, 5.3.1.12, 5.3.1.13, 5.3.2.6 and 5.3.2.9 (see CDRL).

TISSS SOW Task (4.2.3)

Code and test each TISSS software unit in accordance with Attachment No. 1, Section 5.4 (see CDRL). All computer programs developed or assembled under this contract shall be provided to the Government. Computer programs shall be provided in accordance with Attachment No. 3, dated 84 Feb 17, to the Contract.

TISSS SOW Task (4.2.4)

Integrate and test the software units in accordance with Attachment No. 1, Section 5.5 (see CDRL).

TISSS SOW Task (4.2.5)

Verify TISSS operation in accordance with Attachment No. 1, Section 5.6, including 5.6.2.4 (see CDRL).

TISSS SOW Task (4.2.6)

Conduct a configuration management program in accordance with Attachment No. 1, Section 5.7 (in Section 5.7.2.2, delete the words "in accordance with DOD-STD-480") (see CDRL).

TISSS SOW Task (4.2.7)

Implement a Software Quality Assurance Program in accordance with Attachment No. 1, Section 5.8 (see CDRL).

TISSS SOW Task (4.2.8)

Implement planning and control procedures for the software development tasks in accordance with Attachment No. 1, Section 5.9.

TISSS SOW Task (4.2.9)

Coordinate the development of the TISSS with the VHSIC Integrated Design Automation System (IDAS)/VHSIC Hardware Description Language (VHDL) efforts. The contractor shall integrate VHDL into TISSS and shall ensure that TISSS standards are consistent with VHDL. The contractor shall also pursue industry coordination for TISSS standards by participating in industry standards groups and presenting TISSS proposals and perspectives. Participation shall include, as a minimum, the IEEE Standards Coordinating Committee 20 Automatic Test Program Generation Subcommittee and the IEEE Computer Society Design Automation Technical Committee Standards Subcommittee. The contractor shall also pursue coordination with the Electronic Data Interchange Format (EDIF) project.

TISSS SOW Task (4.2.10)

Incorporate two fault simulators into the TISSS system. One shall be the Hierarchical Integrated Test Simulator (HITS), owned by the US Navy, and one shall be a commercially-available fault simulator that shall be selected using the results from the available software review (see paragraph 4.1.2 above). Interfaces shall be prepared and delivered so that both fault simulators are fully interoperable with the TISSS system.

TISSS SOW Task (4.2.11)

Prepare a TISSS Interface Manual to enable others to prepare derivation software for other CAD databases and translation software for other types of AMTE (see CDRL).

TISSS SOW Task (4.2.12)

Validate and verify all software developed and implemented in accordance with the approved Software Test Plan (see paragraph 4.2.1).

TISSS SOW Task (4.2.13)

Provide the capability to validate the adequacy and accuracy of generated TIDB's and test programs for new devices.

TISSS SOW Task (4.2.14)

Prepare validation test suites for each tool identified by the contractor in the Requirements Baseline. These validation test suites shall be limited only by the nonincorporation of the VHDL Simulator. Each validation test suite shall test each critical tool function as identified in the Software Test Plans. Each test suite shall consist of both valid and invalid test cases. Each test suite shall be separately delivered in two units, one for a published validation test suite and one for a reserved Government validation test suite. Both test suites shall be accompanied by manuals that describe their use, proper results, improper results, and pass/fail criteria (see CDRL).

TISSS SOW Task (4.2.15)

(Option). Generate electrical test specifications and electrical test programs for identified VHSIC chips for direct input to the Government Baseline VHSIC tester (GenRad GR-18V configuration installed at RADC as of 1 Sep 86). It is desirable to have chip types from more than one VHSIC fabrication line. Chip types selected shall be primarily random logic as opposed to memory. For at least one selected chip type, demonstrate the TISSS capability to automate the generation of test specifications and automatically generate test programs using both behavioral and structural VHDL models. Using both structural and behavioral models for this chip, verify the stimulus and response correctness of a test vector set supplied by the chip manufacturer and used in the manufacturer's test. Using the structural model for this chip, the contractor shall obtain the fault coverage (via fault simulation) for the same vector set. For any additional chip types selected, the contractor may demonstrate the TISSS capability to generate test specifications and test programs using only

behavioral VHDL models of these chips. Using these behavioral models, the contractor shall verify the stimulus response correctness of a test vector set supplied by the chip manufacturer and used in the manufacturer's test of these chips.

TISSS SOW Task (4.2.16)

Provide one (1) organized training course in system theory and operation upon delivery. This training shall be delivered to a minimum of five, maximum of thirty Government representatives. The training shall be separated according to the level of each attendee (e.g., manager, engineer/computer scientist, technician). This course shall be held at a time and place as specified in the Contract Schedule (see CDRL).

TISSS SOW Task (4.2.17)

The contractor shall deliver, install, and demonstrate the TISSS at RADC, Griffiss AFB, N.Y.

TISSS SOW Task (4.2.18)

(Option). Conduct a six-month "beta-site" program after successful completion of the Functional Configuration Audit and the Physical Configuration Audit. The contractor shall contract with one firm to: use the system with at least two VLSI or VHSIC IC's, identify limitations and failures, submit software change requests, and report all difficulties to both the contractor and to the Government (see CDRL).

Option not selected, no work performed on this task.

TISSS SOW Task (4.2.19)

Prepare a Preliminary and Final Technical Brochure for the TISSS (see CDRL).

TISSS SOW Task (4.2.20)

The contractor shall effect coordination with the U.S. Army ERADCOM "Portability of Test Software" (DAAM-20-84-C-0407) contractor. The TISSS contractor shall become familiar with the results of the Army contract. The TISSS contractor shall participate with the Army contractor in his contract continuation by providing technical information and conducting interchange meetings as required. The TISSS contractor shall consider the recommendations made by the Army contractor. Reporting shall be through the Monthly Status Reports (see CDRL).

TISSS SOW Task (4.2.21)

Provide an Initial Graphics Exchange Specification (IGES) compatible graphics generation and viewing capability and CAD input capability for mechanical drawing.

TISSS SOW Task (4.2.22)

Provide a graphical data base report capability for the data base

management system used by TISSS. Provide the SQL Graph software required to provide this capability.

TISSS SOW Task (4.2.23)

Define the Eclectic Capability for Logistics Information and Product Support for Electronics (ECLIPSE), Operations Concept: The ECLIPSE was previously named the LRM TISSS, but due to confusion with the VHSIC TISSS architecture and funding it has been renamed. Define a computer based system for DoD management of logistic technical data, following Computer-aided Acquisition and Logistics Support (CALS) standards and architecture, for life cycle support of Line Replaceable Modules (LRMs) and Printed Circuit Boards (PCB). Emphasis for this structure, to be defined in the Operational Concept Document and System/Segment Specification shall be on digital, analog, and mixed analog/digital (hybrid) LRMs and PCBs. Major system interfaces shall be defined for product definition input formats for all data or information. Also, define product definition output in applications independent formats for information subsets that are selected for specific applications. Define exchange standards that could be used to transmit the product definitions between dissimilar systems. Define the information flow and information processing requirements of each of the major functions in the U. S. Air Force MASA hardware support concept; reference the Hardware and Software Support Concept attachments to the draft AFR 800-45. Develop operational scenarios, including Avionics Integrity Program (AVIP), integrated diagnostics, failure data collection, environmental information from a Time Stress Measurement Device (TSMD), logistics support analysis records, feedback to the MASA Unified Data Base (MUDB) and MASA Integrated Support Facility (ISF), and LRM and PCB spares procurement, with identification of deployment options, user interaction, human and facility resources, and user skill levels. Specific operational needs shall be identified through a combination of literature searches, telephone contacts, and site visits with prospective users at government, vendor, and contractor facilities. To the maximum extent feasible the system shall make use of VHSIC TISSS tools, data, and data base management tools and concepts. A total support environment shall be defined, not stand alone tools. (See CDRL).

TISSS SOW Task (4.2.24)

Define the ECLIPSE Product Definitions and Exchange Standards: Define the product definition data required at each level of information utilization, consistent with the life cycle support needs of the LRMs and PCBs. For the levels to be addressed, LRM and PCB, all data necessary to accomplish tasks associated with test program set development and integrated diagnostics analysis shall be defined. Define information standards for the MUDB in accordance with MASA objectives, as stated in the MASA Support Requirements Document. Review current U. S. Air Force sponsored efforts for electronic Product Description Exchange Specification (PDES) and determine near term and potential long term applicability to ECLIPSE. These information standards shall include, but not be limited to, the languages required to support test specifications and other LRM products, the environment required to use the languages, and the data model, which includes standards for the format, relationships, and manipulation procedures for the MUDB and CALS Core Specifications (MIL-STD-1840A (Draft)). In addition, interchange formats for importing and exporting to

other systems or tools which are presently defined shall be specified. The MIL-Standards that affect the specification of the information standards shall be identified and evaluated, their potential role in ECLIPSE defined, and any necessary extension or modifications to the MIL-Standards shall be proposed and rationale provided. Attendance at necessary industry and government information and data standards meetings shall be performed. The status and applicability of these and other industry and government standards and formats shall be defined and reported in a standalone volume of the Concept Exploration Technical Report (CETR). In addition, evaluation for ECLIPSE use of these standards and formats shall be made (See CDRL).

TISSS SOW Task (4.2.25)

Define the ECLIPSE Functional Requirements: Derive functional requirements of an ECLIPSE from the operational concept baseline defined in 4.2.23. These functional requirements shall satisfy the needs of the MASA concept for the U. S. Air Force and CALS implementation (See CDRL).

TISSS SOW Task (4.2.26)

VHSIC TISSS Evaluation: Evaluate the VHSIC TISSS concepts to establish the ability to support any of the ECLIPSE operational concepts identified in this study for the LRM and PCB level of electronic integration. Special attention shall be directed at establishing the feasibility of migrating component level design, specification, and test information to support LRM and PCB design, development, qualification, test, and maintenance. The feasibility for utilization of the VHSIC TISSS Data Base (TDB) or an extension of the TDB to satisfy ECLIPSE support needs shall be determined. The VHSIC TISSS system and architecture shall be evaluated for utilization of VHSIC TISSS subsystems in the ECLIPSE, including the VHSIC Hardware Description Language (VHDL) tool set that may be used for design verification in the VHSIC TISSS (See CDRL).

TISSS SOW Task (4.2.27)

Evaluate Tools: Evaluate existing tools, such as analog and digital simulators, and test program generators, for possible integration into the ECLIPSE. These tools may be either government or industry developed. Based on the functional requirements, the contractor shall identify and analyze those software packages that shall satisfy the support and maintenance needs of MASA, including AVIP data analysis tools, if available, and report on their status, availability, ownership, cost, operation, and potential benefit. Any relevant tools shall be evaluated including tools under development by the U. S. Navy, Integrated Diagnostics Support System (IDSS) program. The contractor shall coordinate with the U. S. Navy IDSS Program to ensure that duplication of effort and recommended development overlap is avoided. Additional tools to be evaluated shall be the tools under development by the U. S. Army, the Test Engineers Advisor (TEA) and AI/UUT programs. Also, the applicability of the Automatic Test Equipment Support System (ATESS) tools, currently under development by AFLC, shall be evaluated. Additional tools requiring development will be identified and defined. In addition, the existing AFLC databases for data storage and retrieval, for example REMIS and ATOS, shall be evaluated and a list of each database's inputs, outputs, and functions

will be defined by the other ECLIPSE contractor. Coordination with the other ECLIPSE contractor shall be required to ensure the results of this data base evaluation is integrated into the ECLIPSE tool environment (See CDRL).

TISSS SOW Task (4.2.28)

Evaluate Host Environments: Evaluate the VHSIC TISSS compatible hardware and software required to host the ECLIPSE. Specific attention shall be paid to developing a system that can be easily transported to industry and used within a wide variety of automation environments. Factors to be considered should include user friendliness, software transportability, the types and specifications of host computers required for the development and operation tasks, and run-time libraries. The system organization and communication requirements, e.g. distributed versus centralized database, etc., shall also be addressed (See CDRL).

TISSS SOW Task (4.2.29)

Specify System Requirements: Based upon the defined operational concept baseline, develop the ECLIPSE requirements. Define external interfaces, information objects for database access by tools and information transfer, on-line data flow, define data required to be downloaded from aircraft, identify formats for the information required from the aircraft, and system functions along with ECLIPSE measures of performance (See CDRL).

TISSS SOW Task (4.2.30)

Develop Concept Exploration Technical Report (CETR): The CETR shall contain a list of Government agencies and industry organizations interviewed, minutes of DoD and DoD/industry meetings and reviews, a written description of system functions and characteristics that are recommended for full scale engineering development. In addition, all assumptions made during concept development shall be stated and identified as such. The CETR shall contain the names, addresses, agencies, office symbols, and phone numbers of all people contacted during this effort. The people interviewed shall be listed according to meetings interviewed at and organization affiliation (See CDRL).

TISSS SOW Task (4.2.31)

Coordination with Other ECLIPSE Contractor: Coordinate with the additional ECLIPSE effort contracted by the Government. This coordination shall consist of full and open disclosure of all data gathered by and generated for this ECLIPSE effort. This coordination shall ensure that the other ECLIPSE efforts have inputs into the development of all CDRLs to be developed under this effort. The contractor shall conduct interchange meetings with other available ECLIPSE contractors. Reporting of coordinations and all monthly accomplishments shall be through the Monthly Status Reports (See CDRL).

TISSS SOW Task (4.2.32)

ECLIPSE Insertion Plan: Conceptualize and document a top level plan for ECLIPSE insertion into the USAF Logistics Command (AFLC). This plan will

consist of only identification of AFLC and other Air Force organizations to interface to ECLIPSE. The contractor shall provide a preliminary line of code (LOC) estimate for the ECLIPSE (see CDRL).

TISSS SOW Task (4.3)

The contractor shall conduct the following reviews at times and places as specified in the Contract Schedule. These shall be in accordance with Attachment No. 2, MIL-STD-1521B (USAF) "Technical Reviews and Audits for Systems, Equipment, and Computer Software", with the exception of section .4, 5.1.2, 5.1.9, and 5.1.11. The offerer may propose changes in the review process in accordance with Section 100.4.3.

TISSS SOW Task (4.3.1)

(Phase I) One kick-off meeting. This meeting shall be held for the purpose of familiarization of personnel, review and discussion of technical issues (see CDRL).

TISSS SOW Task (4.3.2)

(Phase I) One Preliminary Specification Review (PSR). This review shall be held for the purpose of reviewing progress and receiving technical comments from the Government during the design phase (see CDRL).

TISSS SOW Task (4.3.3)

(Phase I) One interim design phase presentation (see CDRL).

TISSS SOW Task (4.3.4)

(Phase II) Two Critical Milestone Reviews (CMR). These CMR's shall be held for the purpose of reviewing progress and receiving technical comments from the Government (see CDRL).

TISSS SOW Task (4.3.5)

(Phase II) Quarterly interim progress reports (see CDRL).

TISSS SOW Task (4.3.6)

(Phase II) Two open reviews for industry, Government, and academic personnel. The first shall be held in an Eastern US location twelve months after Phase II award. The second shall be held two months prior to the Functional Configuration Audit in a location of the contractor's choice.

TISSS SOW Task (4.3.7)

(Phase II) A Functional Configuration Audit and Physical Configuration Audit to be held at RADC upon completion of the installation of TISSS.

TISSS SOW Task (4.3.8)

(Phase II) A final presentation within two weeks of contractor's receipt of Government comments on the final technical report (see CDRL).

TISSS SOW Task (4.3.9)

Kick-off Meeting. This meeting shall be held for the purpose of familiarization of Government personnel (all services). Review and discussion of technical issues associated with the ECLIPSE study effort shall be conducted (See CDRL).

TISSS SOW Task (4.3.10)

Technical Interchange Meeting (TIM): Conduct a TIM with Government and industry to present the status of technical and programmatic progress made to date in performance of the ECLIPSE study effort. The presentation will be given to approximately two hundred (200) Government and Industry personnel and shall not last more than three (3) days. One day of the TIM shall be used to present preliminary developed DoD CALS product definitions and exchange standards to industry and Government participants for comment (See CDRL).

TISSS SOW Task (4.3.11)

LRM Standards Meeting Participation. Participate in a DoD and industry review of proposed standards for LRMs to be performed by the USAF ASD contractor. One technical representative shall be present. The status of technical and programmatic progress made to date in the performance of the ECLIPSE study effort shall be presented (See CDRL).

TISSS SOW Task (4.3.12)

System Requirements Review (SRR). Conduct an SRR seven months after award. This review shall be held in accordance with MIL-STD-1521B. The Operational Concept and System Segment Specification document contents shall be presented. The presentation will be given to approximately two hundred (200) Government and industry personnel and shall not last more than four (4) days. One (1) day of the SRR shall be used to present developed DoD CALS product definition and exchange standards to industry and Government for comment (See CDRL).

TISSS SOW Task (4.3.13)

Final Government Meeting. Conduct a final Government meeting in conjunction with the SRR. This meeting shall be held on the last day of the SRR to familiarize Government representatives with the results and future impacts of the ECLIPSE concept exploration (see CDRL).

TISSS SOW Task (4.4) (Option).

Develop the VHDL/TVL Validation Software in accordance with Attachment No. 5 to the Statement of Work. Deliver, install, and demonstrate this stand alone simulation validation software on the TISSS host computer at RADC. All software developed under this task shall be in full accordance with Attachment No. 1 (DoD-STD-SDS) except where explicitly deleted in the SOW.

APPENDIX B
TISSS FOUR COLOR BROCHURE

(NOT INCLUDED)

APPENDIX C

SELECTION OF COMPUTER SOFTWARE CONFIGURATION ITEMS (CSCIS)

The following is extracted from MIL-STD-483A, 4 Jun 85, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, Appendix XVII, Criteria for Selecting Configuration Items. "170.7 Effects of Selecting Too Many Configuration Items. Too many configuration items may result in effects hampering visibility and management rather than improving it. These effects include:

1. Increased administrative burden in preparing, processing, and status reporting of engineering changes which tends to be multiplied by the number of configuration items.
2. Increased development time and cost as well as possibly creating an inefficient design.
3. Possible increase in management effort, difficulties in maintaining coordination and unnecessary generation of paper work."

The table below uses the questions to be asked and criteria for evaluating the answers found in MIL-STD-483A paragraph 170.9 Configuration Item Selection Checklist. Each question is answered for each TISSS "CSCI" and evaluated for results. Conclusion: TISSS should be two CSCIs.

Question	U	C	D	A	O	A	A	M	V	H	H
	I	P	I	U	U	P	D	I	S	S	L
a. Is it critical high risk, and/or a safety item?	N	N	N	N	N	N	N	N	N	N	N
b. Is it readily identifiable with respect to size, shape, and weight (hardware)?	N	N	N	N	N	N	N	N	N	N	N
c. Is it newly developed?	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
d. Does it incorporate new technologies?	N	N	N	N	N	Y	N	N	Y	N	N
e. Does it have an interface with hardware or software developed under another contract?	Y	N	N	N	N	N	Y	Y	Y	Y	Y
f. With respect to form, fit or function, does it interface with other configuration items whose configuration is controlled by other entities?	Y	N	N	N	N	N	Y	Y	Y	Y	Y
g. Is there a requirement to know the exact configuration and status of changes to it during its life cycle?	N	N	N	N	N	N	N	N	N	N	N
	U	C	D	A	O	A	A	M	V	H	H
	I	P	I	U	U	P	D	I	S	S	L

- | | |
|---|---|
| 1. Number of Yes answers. | 3 1 1 1 1 2 3 3 4 2 2 |
| 2. Did you answer Yes to more than 50% of the questions? | N N N N N N N N Y N N |
| 3. According to MIL-STD-483A, should this CSCI have been designated a CSCI? | N N N N N N N N Y N N |

APPENDIX D

PATENT APPLICATION

TITLE: A Process for the Generation of Test Programs from Tester
Independent Test Specifications

INVENTORS: L. Shombert, R. Rolfe, J. Garnett

This disclosure covers the process whereby the TISSS Postprocessor translates a TISSS test specification into a GR-18 test program. The Postprocessor has already demonstrated its flexibility in adapting to the needs of Components Engineering and all the evidence indicates that it will be a very powerful tool once TISSS has been integrated into the test environment. No other comparable function is available from external suppliers. Test generation software from tester manufacturers tends to be inflexible and is, of course, targeted at a single tester.

A Process for the Generation of Test Programs from Tester Independent Test Specifications

D.1 POSTPROCESSOR OPERATION - INTRODUCTION

The purpose of the TISSS Postprocessor is to convert a TISSS test specification, which is independent of any particular tester, into a tester specific test program. The current TISSS Postprocessor is targeted at the GenRad GR-18. The Postprocessor data flow is shown in Figure 1. The inputs are the tester independent test specification, tester specific (but device independent) template code, and tester specific, site specific configuration files.

The Postprocessor generates a test program by altering template code, which is basically boilerplate test code, according to the data contained in the test specification and the configuration files. This disclosure is concerned with the process whereby the template code is altered.

D.2 POSTPROCESSOR OPERATION - THEORY

The key to Postprocessor operation is the input template code. Template code consists of plaintext interspersed with directives. Directives are identified by a prefix character and obey a defined syntax. Plaintext is any sequence of zero or more characters that does not contain the directive prefix character.

The Postprocessor expands template code by copying plaintext verbatim to the output until a directive is reached. Directives are not copied to the output but are treated as instructions to the Postprocessor. However, a directive may generate text that is written to the output. A directive may also result in a change in control of the Postprocessor, for example, causing the Postprocessor to read from a different piece of template code, or write to a different output. Once the directive has been executed, the Postprocessor resumes copying plaintext to the output, waiting for another directive. This process is shown in Figure 2 in flowchart form.

The Postprocessor is able to generate a test program because some directives operate on data from the test specification. Some directives allow this information to be inserted into the output, while others guide Postprocessor expansion. For example, a directive might cause a voltage from the test specification to be written to the output. The surrounding plaintext, when processed as a test program, might instruct a tester to set a power supply to the test specification voltage.

The most important characteristic of the Postprocessor is that there are no restrictions placed on the plaintext. Furthermore, directives that generate output generate the smallest possible amount of output, so as to reduce the bias towards a particular format or syntax for the output. As a result, the Postprocessor can generate GR-18 test programs, tabular reports, and VAX VMS DCL command procedures with equal ease.

D.3 POSTPROCESSOR OPERATION - IMPLEMENTATION

In the current TISSS Postprocessor, directives consist of the prefix character "#" followed by a name and a set of parenthesis surrounding a (possibly empty) list of arguments. A special type of directive is the token, which consists of the prefix character "#" followed by a set of parenthesis surrounding a name. A third type of directive is the prefix insertion directive which is the string "##".

Examples of directives are:

```
#temperature()  
#config(FIXTURE, BOARD_ID)  
#(VOLTAGE.VALUE)  
##
```

Note that all directives are immediately identifiable by the prefix character. Furthermore, it is possible to exactly determine the end of the directive, either a close parenthesis or, in the case of the prefix insertion directive, the second "#". The important point is that a directive may be extracted from template code regardless of the surrounding plaintext.

The tester independent test specification is written in the TISSS Test Description Language (TDL). In essence, a test specification in TDL is a sequence of test requests (called test macros) that resemble procedure calls in high level languages. An example test macro is shown in Figure 3. Note that it is a list of parameters, each of which is assigned a value in the test specification.

A test macro represents a particular test method, such as a means of testing power supply current. The Postprocessor, under control of a master template, associates test macros from the test specification with test macro templates from a template library. The test macro template plaintext contains the test method, and directives in the test macro template "customize" the output to perform the test according to the values assigned to the various test macro parameters. A possible test macro template for the test macro of Figure 3 is shown in Figure 4, and the output, if run through the Postprocessor is shown in Figure 5. Only token directives are shown, which extract data values from the test specification.

The TISSS Postprocessor provides directives that allow a template to step through the test plan (control when the next test macro is processed). The Postprocessor also provides directives to conditionally expand template code. This is important when different code must be expanded for different ranges of test macro parameter values. Other directives are provided to manipulate tester resources.

D.4 POSTPROCESSOR OPERATIONS - EXTENSIONS

The following extensions are not implemented in the current TISSS Postprocessor, but are part of the overall concept:

1. This process is not limited to text, although textual templates are the easiest to handle. The process could handle, for instance, 8-bit bytes, 6-bit bytes, 16-bit words. The process can even be built around bit strings, in which case the input and output would be binary files.
2. This process is not limited to the functions (directives) specified for the TISSS Postprocessor.
3. This process includes the capability to create and modify local variables from within the Postprocessor, and to make decisions based on those variables. For instance, a variable named TEST_COUNT could be declared and set to zero, and then incremented whenever a new test was generated. Whenever TEST_COUNT reached a multiple of 10, a special template would be processed.
4. This process includes the capability to allow a user to define new directives. Such a capability could be developed using a Lisp-like interpreter.
5. This process includes the capability to translate source other than TISSS Test Description Language, however, the source must follow the TDL structure of high-level function calls inside nested loops.

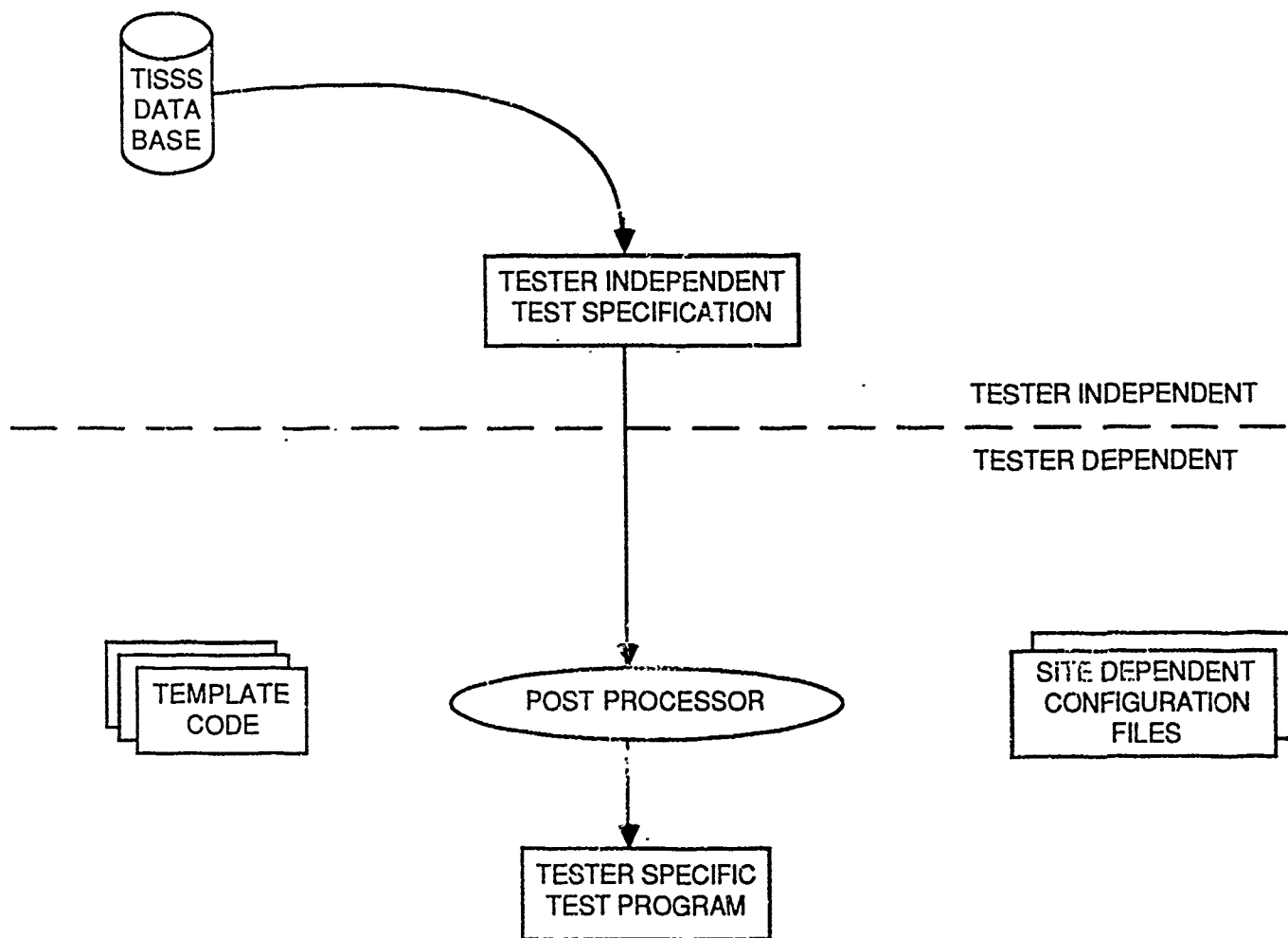


FIGURE C-1 POSTPROCESSOR DATA FLOW

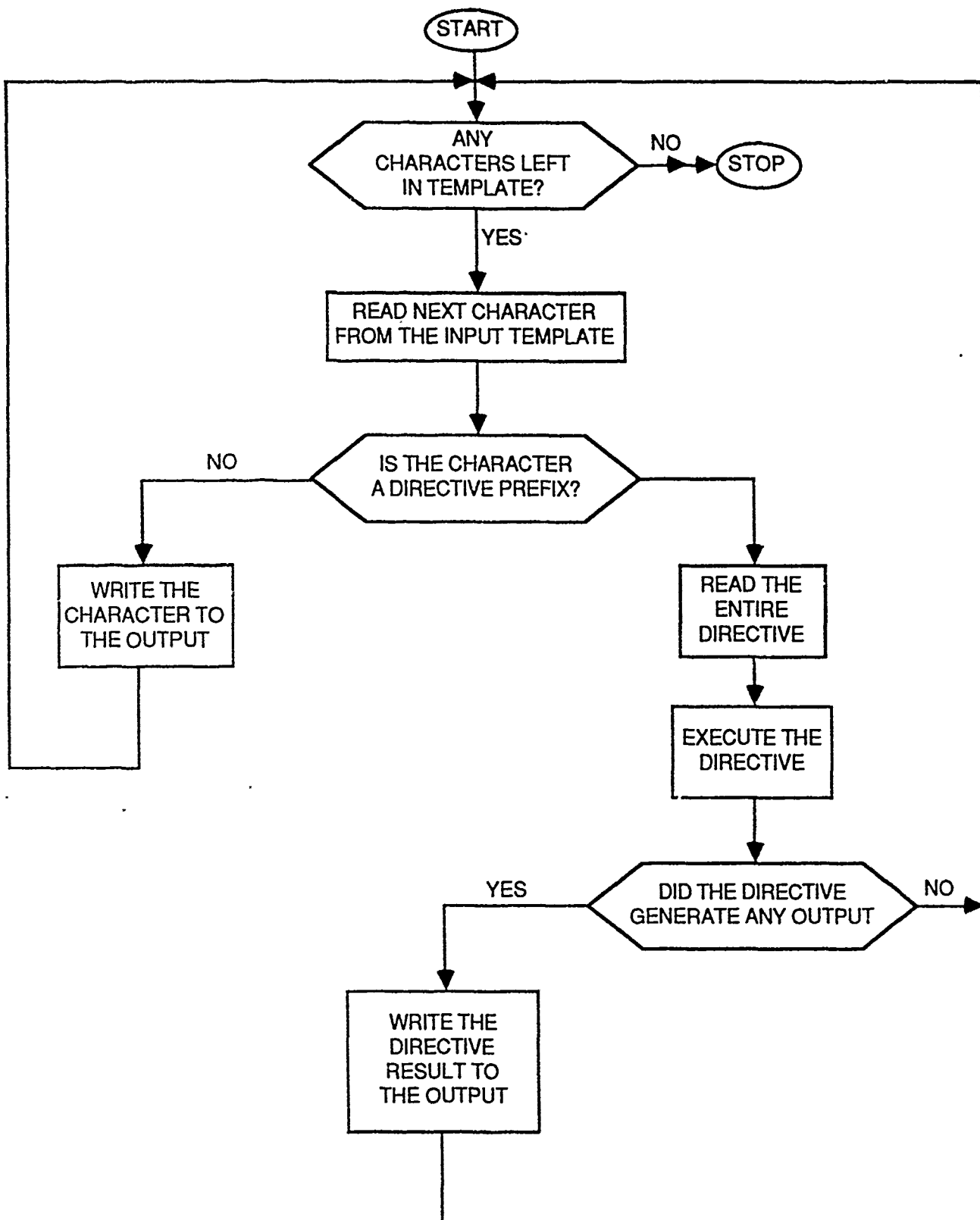


FIGURE C-2 TEMPLATE EXPANSION FLOW CHART

```

test_macro A_TEST_MACRO
(SYMBOL      :_STRING      => "Any string ",
 LOW_VOLTAGE : VOLTAGE_VALUE =>
  (_VALUE      => 0.0_V,
   ACCURACY    => 0.1 V),
 HIGH_VOLTAGE : VOLTAGE_VALUE =>
  (_VALUE      => 5 V,
   ACCURACY    => 0.1 V),
 COUNT       : POSITIVE => 5);

```

FIGURE 3
Sample test macro

```

-- !!#(symbol)!!
for i in [1..#(COUNT)] loop
  PUT((#(HIGH_VOLTAGE.VALUE) - #(LOW_VOLTAGE.VALUE)) / #(COUNT)) * i)
end loop;

```

FIGURE 4
Sample test macro template

```

-- !!Any string !!
for i in [1..5] loop
  PUT((5V - 0.0V) / 5 * i);
end loop;

```

FIGURE 5
Expanded test macro template